Antti Toura & Rasmus Pettersson

# Reaaliaikaisten Kalman- ja Madgwick -suodinten vertailu kolmiulotteisessa tilapaikannuksessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

7.10.2013

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Antti Toura, Rasmus Pettersson<br>Koura -stabilointijärjestelmä<br><br>84 sivua + 2 liitettä<br>7.5.2015 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Tietotekniikka |
| Suuntautumisvaihtoehto | Sulautettu tietotekniikka |
| Ohjaaja(t) | Anssi Ikonen |

Koura on projektinimi yleiskäyttöiselle stabilointijärjestelmälle. Kouran ensimmäinen prototyyppi toteutettiin Teknologia'13 -messuilla käytyyn Elektroniikkainsinöörien seuran järjestämään Wärkkäyskilpailuun yhdeksi kilpailuprojektiksi.

Stabilointijärjestelmän tarkoitus on vakauttaa kappaletta, ja se on suunniteltu mahdollisimman yleiskäyttöiseksi. Vakautus toimii lukemalla kappaleen asentoa sensoreilla, ja korjaamalla mahdolliset ulkopuolisesta liikkeestä johtuvat kääntymiskulmat moottoreiden liikkeellä. Näin kappale pysyy suorassa, vaikka jokin ulkopuolinen voima yrittäisi kallistaa kappaletta.

Järjestelmä toteutettiin PSoC4 alustalle. Prosessorina toimii ARM Cortex M0. Lisäksi lisättyjä sensoreita ovat MPU6050 IMU, jossa on 3-akselinen gyroskooppi, sekä 3-akselinen kiihtyvyysmittari joita käytettiin kappaleen asennon määrittämiseen. Kappaleen asennon korjaaminen toteutettiin servomoottoreilla.

| | |
|---|---|
| Avainsanat | PSoC 4, IMU, Gyroskooppi, Kiihtyvyysanturi, Stabilointi, Kalman, PID, MPU6050, Wärkkäyskilpailu, Teknologia '13 |

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Antti Toura, Rasmus Pettersson<br>Koura -stabilization system<br><br>84 pages + 2 appendices<br>7.5.2015 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Specialisation option | Embedded systems |
| Instructor(s) | Anssi Ikonen |

Koura is the project name for a global stabilization system. The first prototype was created as an entry to the Wärkkäyskilpailu electronics and programming design contest. The contest was held at the Teknologia'13 electronics faire.

The purpose of the stabilization system is to stabilize any object and is designed to be as global as possible. The stabilization works by reading the position of the object and reacting accordingly to possible outside forces that might rotate the object to an undesired position.

The system was implemented using a PSoC4 evaluation board. The processor is ARM Cortex M0. Additional sensors included MPU6050 gyroscope and an accelerometer as well as an HMC5883l magnetometer that were used to track the objects position. Servo motors were used for the counter movements on the object.

| | |
|---|---|
| Keywords | PSoC 4, IMU, Gyroscope, Accelerometer, Stabilization, Kalman, PID, MPU6050, Wärkkäyskilpailu, Teknologia '13 |

**Sisällys**

## Lyhenteet

| | |
|---|---|
| PSoC | Mikrokontrollerialusta / kehitysalusta. Lyhenne sanoista programmable system on chip. |
| MARG | Magnetic, Angular Rate, and Gravity. Järjestelmä, joka on tarkoitettu kappaleen asennon määritykseen. MARG -järjestelmissä on 9 vapausakselia. |
| AHRS | Attitude and heading reference system. Järjestelmä, joka on tarkoitettu kappaleen asennon määrittämiseen. |
| IMU | Lyhenne sanoista ”Inertial Measurement Unit.” Kappaleen liikkeen mittaukseen tarkoitettu piiri. Koostuu gyroskoopista sekä kiihtyvyysanturista (6 vapausakselia). |
| PWM | Pulse Width Modulation. Kellopulssin pituuteen nojaava tiedonsiirtomenetelmä. |
| PID | Proportional-integral-derivative-säädin. |
| MEMS | Mikrosysteemi. Lyhenne sanoista ”Micro Electrical Mechanical System.” |
| UART | Sarjaliikenneprotokolla. Lyhenne sanoista ”Universal Asynchronous Receiver Transmitter.” |
| USB | Universal Serial Bus. Sarjaliikenneprotokolla. |

# 1 Johdanto

Tässä opinnäytetyössä käydään läpi teollisuuden yleiskäyttöisimpiä suotimia tilamääritykseen, sekä toteutettu statistinen data-analyysi kahden eri suodinalgoritmin välillä. Suotimiksi valikoitiin Kalman- sekä Madgwick -suotimet. Sen lisäksi työssä käsitellään teoriaa PID -säätimestä sekä komplementaarisuotimesta. Suotimet toteutettiin myös kahteen eri käyttökohteeseen, jotta voitiin demonstroida suotimien käyttöä konkreettisissa käyttökohteissa. Käyttökohteeksi toteutettiin yleiskäyttöinen alustasta riippumaton stabilointi- sekä tilamääritysjärjestelmä. Stabilointijärjestelmän kulmakivenä on kappaleen tilan määritys kolmiulotteisessa tilassa, joten myös inertiapaikannuksen teoria käydään läpi kattavasti. Tässä työssä esitelty yleiskäyttöinen stabilointijärjestelmä on ollut voittajalaite vuonna 2013 järjestetyssä Elektroniikkaliiton Wärkkäys'13 -kilpailussa. Opinnäytetyön ehdottomassa keskiössä on näiden suodinalgoritmien data-analyysin lisäksi sovellutuksena toteutettu mikropiiri, jolla pystytään havainnoimaan kappaleen asento kolmiulotteisessa tilassa, sekä tämän mikropiirin jatkosovellutukset.

Stabilointijärjestelmiä käytetään monella teollisuuden alalla. Varsinkin auto-, laiva-, vene-, sekä ilmailuteknologiassa teknologia on erittäin tärkeää, mutta käyttökohteita löytyy myös valokuvauksessa sekä arkipäiväisemmissä vakauttamistarpeellisissa teknologioissa. Vakauttamisjärjestelmän keskustassa toimii yleisesti ottaen gyroskooppi, sekä kehittyneemmissä järjestelmissä sen yhteyteen on liitetty myös kiihtyvyysanturi sekä elektroninen kompassi. Gyroskooppi ja kiihtyvyysanturi sekä niiden käyttö ovat tämän päivän teknologiassa erittäin ajankohtaisia, sillä itsensä, sekä oman paikkansa tiedostavien laitteiden kysyntä markkinoilla kasvaa jatkuvasti. Tämä sensoriyhdistelmä löytyy jo monesta arkipäivän laitteesta mm. suurimmasta osasta älypuhelimista sekä muista älykkäistä mobiililaitteista. Tämä projekti antaakin erinomaiset lähtökohdat gyroskooppien ja kiihtyvyysantureiden datan keräämiseen ja saadun datan käyttöön erilaisissa teknisissä sovellutuksissa.

Stabilointijärjestelmän tarkoitus on vakauttaa kappaletta, ja se on suunniteltu mahdollisimman yleiskäyttöiseksi. Vakautus toimii lukemalla kappaleen asentoa sensoreilla, ja korjaamalla mahdolliset ulkopuolisesta liikkeestä johtuvat kääntymiskulmat moottoreiden liikkeellä. Näin kappale pysyy suorassa, vaikka määrittelemätön ulkopuolinen voima yrittäisi kallistaa kappaletta.

Järjestelmä toteutettiin PSoC4-alustalle. Prosessorina toimii ARM Cortex 0. Lisäksi lisättyjä sensoreita ovat MPU6050 IMU, jossa on 3-akselinen gyroskooppi, sekä 3-akselinen kiihtyvyysmittari.

# 2 Kappaleen asennon määritys kolmiulotteisessa tilassa

Tämän insinöörityön käytännön osuuden ymmärtämiseksi on olennaista ymmärtää teoreettinen pohja kappaleen asennon esitysmuodoista. Kappaleen asennon määritys kolmiulotteisessa tilassa onkin yksi tilapaikannuksen kulmakiviä. Jotta järjestelmä pystyy ymmärtämään sen omaa asentoa, tulee olla olemassa matemaattinen esitysmalli kappaleen asennosta eri ulottuvuuksissa, esitys mahdollisista liikkeistä sekä kääntymisistä tässä tilassa. Tämän lisäksi samoja matemaattisia malleja käytetään hyvin paljon mm. kolmiulotteisessa mallintamisessa sekä tietokonegraafikoissa. Matemaattisia malleja kappaleen määrittämiseen on kehitetty jo useita, jokainen omine vahvuuksineen sekä heikkouksineen. Tässä kappaleessa esitetään yleisimmät matemaattiset esitysmuodot.

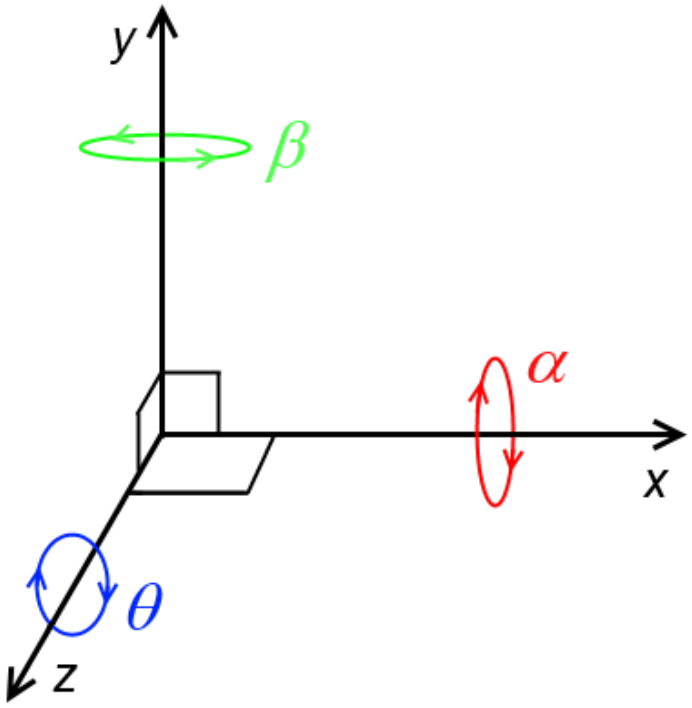
## 2.1 Eulerin kulmat

Eulerin kulmilla tarkoitetaan matemaattista esitysmuotoa jäykän kappaleen tai vektorin asennosta kolmiulotteisessa tilassa. Eulerin kulmat koostuvat kolmesta kulmasta kolmen kuvitteellisen akselin suhteen. Näitä akseleita voidaan kutsua nimillä $x$, $y$ ja $z$. Nämä symboliset nimitykset ovat yleisesti käytössä puhuttaessa Eulerin kulmista sekä kolmiulotteisista koordinaatistoista. Akselien suunnat vastaavat myös yleisesti käytössä olevia koordinaatistovastineita, missä $y$ kuvaa vertikaalista akselia ja $x$ ja $z$ kuvaavat kahta jäljelle jäänyttä horisontaalista akselia.

Eulerin kulmat kuvaavat kappaleen asentoa, käyttämällä esitysmuotoa, jossa kappaleen asento esitetään kolmena kulmana, jotka kuvaavat kiertymää näiden akselien ympäri. Yleisesti näistä kiertymistä käytetään englanninkielisiä nimityksiä *yaw*, *pitch* ja *roll* [5]. Suomenkielisinä nimityksinä käytetään kallistus, pituuskallistus ja suuntakulma. Suuntakulma kuvastaa kiertymää *y*-akselin ympäri, kallistus kuvastaa kiertymää *x*-akselin ympäri ja pituuskallistus kuvastaa kiertymää *z*-akselin ympäri. Yleisessä käytössä ovat myös kulmamerkinnät $\alpha$, $\beta$ sekä $\theta$. Muita yleisessä käytössä

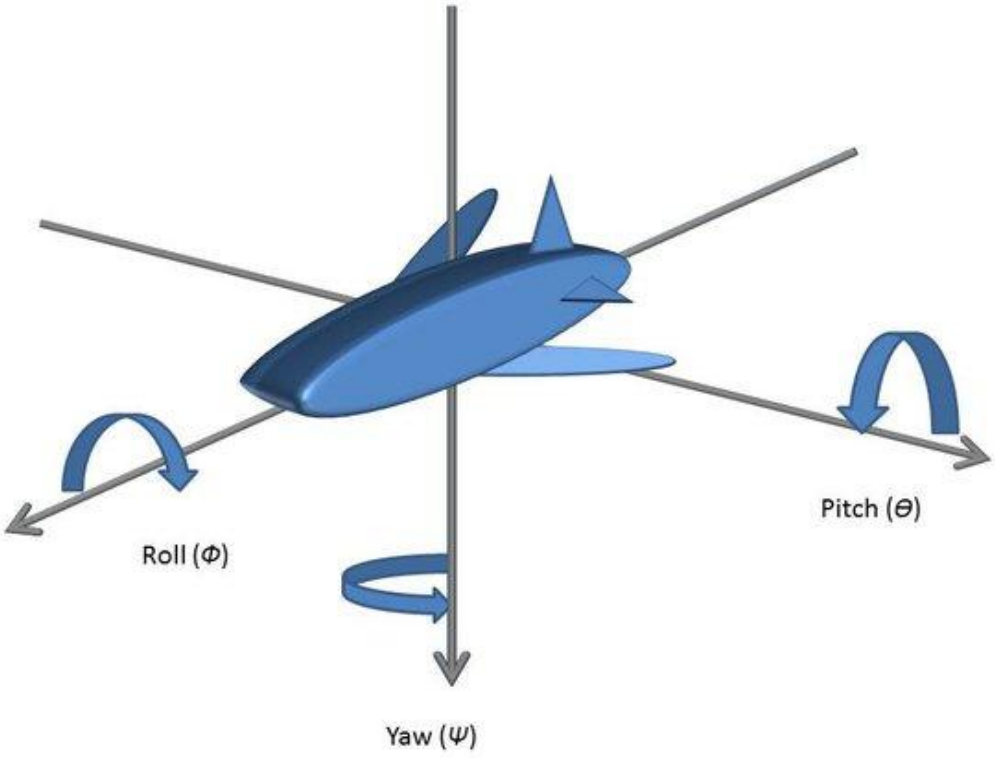
olevia nimityksiä ovat *heading*, *attitude* ja *bank*, sekä *azimuth*, *elevation* ja *tilt*, noudattaen edellä mainittua järjestystä. [3]

Helpoin tapa havainnollistaa tätä koordinaatistoa sekä kiertymää akselien ympäri on käyttämällä kuvaajaa. Koordinaatisto sekä niitä vastaavat akselien kiertymät on kuvattu kuviossa 1.




Kuvio 1. Koordinaatisto sekä akseleita vastaavat kiertymät.

Kappaleen asento tai yksikkövektori voidaan siis esittää yhdistämällä nämä kolme kiertymää yhdeksi esitysmuodoksi. Eulerin kulmia voidaan ajatella myös liikkeenä. Tällöin liike kuvataan kolmen eri sarjassa tapahtuvan liikkeen yhdistelmänä. Tarkoittaen, että ensin käännytään kulma $\alpha$ akselin $x$ suhteen, sitten kulma $\beta$ $y$-akselin suhteen, ja viimeisenä kulma $\theta$ $z$-akselin suhteen. Myös tätä kappaleen asentoa on helpoin havainnollistaa kuvaajalla, ja se on havainnollistettu kuviossa 2.



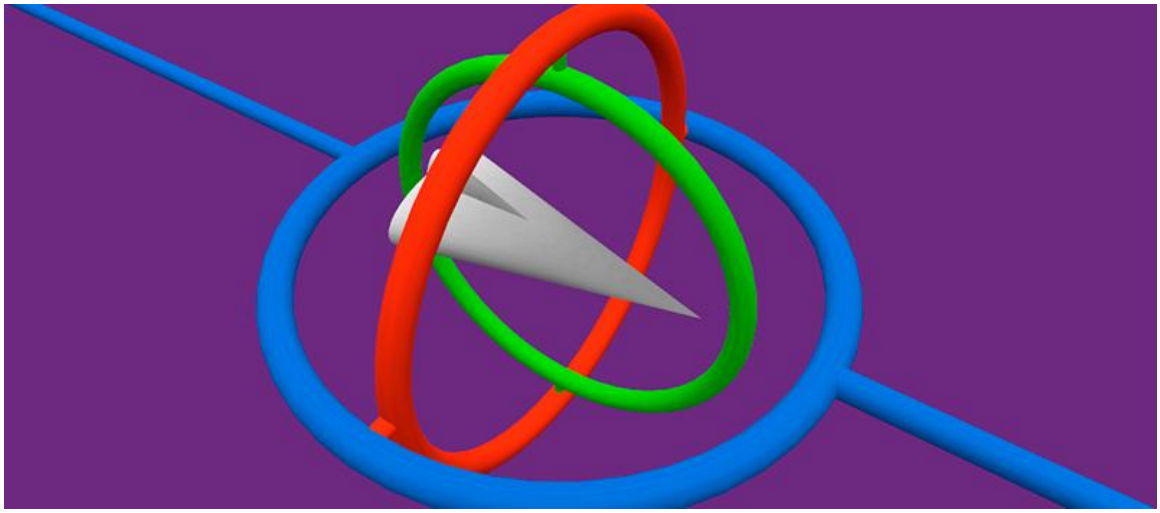

Kuvio 2. Kappaleen esittäminen Yaw, pitch & roll -muodossa. [13]

Käytännön työssä törmätään hyvin nopeasti kuitenkin Eulerin kulmien ongelmiin sekä rajoituksiin. Ensimmäinen ongelma Eulerin kulmien käytössä on se, että minkäänlaisia standardeja esitymuodosta tai kulmien järjestyksestä ei ole määritelty. Tämä tarkoittaa, että mikäli kulmat annetaan pelkkänä lukuna, voi havaitsija erehtyä kulmien järjestyksestä, ja näin ollen luoda täysin väärän kuvan kappaleen asennosta. Esimerkkinä annettakoon, että yleisin esitysmuoto kulmien järjestykselle on x-y-z, mutta yleisiä permutaatioita ovat myös y-z-x sekä y-x-z. Erilaisia yhdistelmiä kulmien esitykseen on siis kuusi. Tämän lisäksi ei ole myöskään määritelty yhtenäisesti, mihin akseliin kulmat ovat sidottuja. Yaw, pitch ja roll kulmien akselit saattavat vaihdella, vaikkakin yleisin esitystapa on kuvattu kuviossa 2. Tämän lisäksi myöskään rotaatioiden pyörimissuuntia ei ole määritelty yhtenäisesti. [3]

Eulerin kulmien matemaattisiin ongelmiin kuuluu, että millekään kappaleen asennolle ei ole olemassa ainoastaan yhtä Eulerin kulmien esitysmuotoa. Mikä tahansa asento voidaan ilmaista myös eri yhdistelmällä kulmia, kuin mikä ehkä olisi yksinkertaisin esitysmuoto. Tämä tulee ottaa huomioon erityisesti käytännön sovelluksissa, kuten ohjelmakoodissa.

Eulerin kulmilla on erittäin yksinkertaista ja innovatiivista esittää staattisen kappaleen asentoa tai staattista vektoria. Sen sijaan kappaleiden liikettä on vaikea kuvata Eulerin kulmilla. Kuvitellaan tilanne, jossa alkutilanteena pidämme vektoria kuvitteellisessa kulmassa *{a, a, a}*. Pystymme lisäämään rotaatioon uuden rotaation *{b, b, b}*. Nämä kaksi rotaatiota yhdistettynä antavat vektorillemme uuden asennon, jota voimme kutsua muuttujana *{c, c, c}.* Ei kuitenkaan ole olemassa tapaa kuvata itse rotaatiota kohteeseen *{c, c, c}*. Emme voi kuvata lopputulosta pelkästään laskemalla yhteen vektoreita *{a+b, a+b, a+b}*, sillä käännösten suoritusjärjestyksellä on merkitystä käytettäessä Eulerin kulmia. Samat käännökset toteutettuna eri järjestyksessä, saattavat asettaa vektorin täysin eri asentoon.

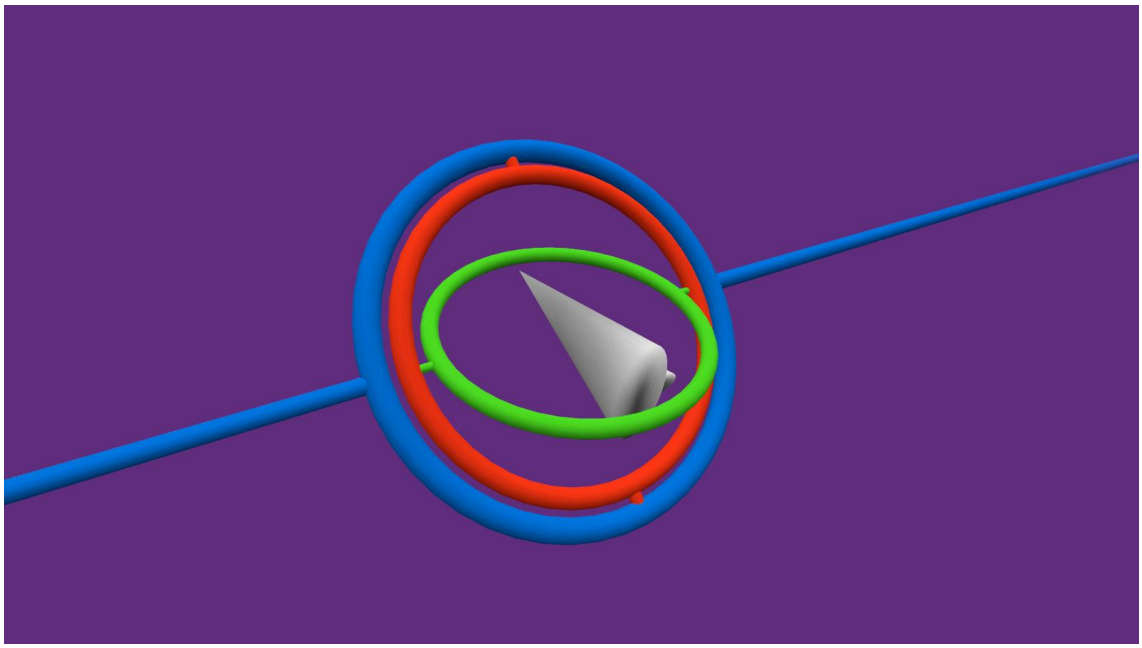
Kappaleen rotaatio akseleilla voidaan kuvitella myös kehänä kappaleen ympärillä. Kehä on staattinen kappaleen suhteen, tarkoittaen, että mikäli kehää käännetään, kääntyy myös kappale. Mikäli käännetään ulompaa kehää, vaikuttaa se myös sisempien kehien asentoon. Näitä kehiä nimitetään *Gimbaleiksi* tai *Gimbal* -akseleiksi. Gimbalit on havainnollistettu kuviossa 3. Helpoin tapa kuvitella gimbaleita on kuvitella vanhanaikainen mekaaninen gyroskooppi.



Kuvio 3. Gimbal -akselit. Kuvasta voidaan havainnoida akselien vuorovaikutus toisiinsa. [12]

Ehkä suurin yksittäinen ongelma Eulerin kulmissa on se, että ne kärsivät ilmiöstä nimeltä *gimbal lock* [24]. Gimbal lock -tilanteita syntyy, kun kaksi akselia asettuu toistensa kanssa linjaan. Käytännössä tilanne esiintyy joka kerta, kun kappaletta käännetään 90º minkä tahansa yhden akselin ympäri. Näissä kohdissa esiintyy

erikoispiste eli singulariteetti, jossa rotaatio yhden akselin ympäri, vaikuttaa suoraan sen linjassa olevan akselin rotaatioon, kuten havainnollistettu kuviossa 4. Mikäli kuvassa havainnollistetussa tapauksessa käännettäisiin sinistä gimbalia, kääntyisi myös punainen gimbal sinisen gimbalin rotaation verran.



Kuvio 4. Gimbal lock -tilanne. [12]

Tilanteessa menetetään siis yhden ulottuvuuden vapaa liike. Gimbal lock voi aiheuttaa teknisissä sovelluksissa arvaamatonta käytöstä saavuttaessa singulariteetteihin. Ongelma onkin ollut menneisyydessä vastuussa mm. lentokoneiden putoamisesta sekä ongelmista Apollo 11 kuulennolla [8]. Ongelman yleisimpiin ratkaisuihin kuuluu neljännen gimbalin eli akselin lisääminen. Toinen yleinen ratkaisu on välttää gimbaleiden käyttöä kokonaan siirtymällä rotaatioesityksessä esitysmuotoihin, jotka eivät pohjaa akseleihin, kuten kvaternioihin tai rotaatiomatriiseihin. Tämä ei kuitenkaan ole mahdollisuus systeemeissä, jotka pohjaavat Eulerin kulmiin.

Ongelmia Eulerin kulmien kanssa saattaa myös esiintyä, kun pyörähdetään täysi ympyrä minkä tahansa akselin ympäri. Kappaleen ollessa asennossa 360º, on se samaan aikaan asennossa 0º. Tämä tulee ottaa ohjelmakoodissa huomioon, jotta ohjelmaan ei kehity tilaa, jossa kappaleen asentoa ei ole määritelty. Vaikka tämä tila olisikin määritelty, saattaa esiintyä ongelmia siirryttäessä yhtäkkiä asteesta 359 asteeseen 0. Mikäli tilannetta ei huomioida käytännön ohjelmakoodissa, saattaa se

aiheuttaa arvaamatonta käytöstä ohjelmassa, mikä taas voi aiheuttaa vaurioita fyysisissä järjestelmissä. [3]

Tämän lisäksi rotaatioiden kuvaaminen Eulerin kulmilla on ongelmallista. Koska rotaatiot Eulerin kulmissa esitetään yhden asennon muutoksesta toiseen, ja koska on otettava huomioon aina gimbaleiden suoritusjärjestys, saattavat rotaatiot yleensä vääristyä. Esitetty rotaatio ei aina ole suorin rotaatio tilanteesta A tilanteeseen B.

Eulerin kulmien etuihin taas kuuluu niiden helppous sekä innovatiivisuus. Eulerin kulmat ovat tilaesitysmuodoista helpoin omaksua. Ne ovat helpoimmat kuvitella mielessään, kun havainnoitsijalle annetaan pelkät Eulerin kulmien arvot. Käytettäessä muita muotoja, kuten suuntamatriiseja tai kvaternioita, on pelkistä lukuarvoista melkein mahdotonta kuvitella kappaleen todellista asentoa. Eulerin kulmat sopivat siis erittäin hyvin kappaleen asennon esittämiseen, mutta törmäävät edellä mainittuihin ongelmiin, kun niillä yritetään kuvata rotaatioita.

## 2.2 Kvaterniot

Kvaternio on neliulotteinen kompleksiluku, jolla voidaan kuvata jäykän kappaleen asentoa kolmiulotteisessa tilassa, mutta myös rotaatioita. Kvaternioilla on myös laajalti sovellutuksia käytännön ja teoreettisen matematiikan saralla [5]. Siinä missä normaali kompleksiluku $t + xi$ koostuu reaaliosista $x$ ja $w$ sekä imaginääriosasta $i$, on kvaterniossa kolme imaginääriosaa $i$, $j$ ja $k$. Kvaterniot noudattavat siis muotoa

$$w + xi + yj + zk$$

Missä $t$, $x$, $y$ ja $z$ ovat reaalilukuja ja yllä mainitut $i$, $j$, ja $k$ ovat imaginäärilukuja. Sir William Hamiltonin kehittämä peruskaava kvaternioille on määritetty seuraavasti

$$i^2 = j^2 = k^2 = ijk = -1$$

Kvaterniot voidaan kuvitella myös yhden reaaliluvun ja kolmiulotteisen vektorin yhdistelmäksi. Tämä ajattelumalli saattaa helpottaa kvaternioiden ymmärtämistä tilapaikantamisessa. Tässä mallissa on olemassa siis kolme imaginääriakselia vektorille, sekä yksi reaaliakseli, jota voidaan kuvata tasona. Kaikki suoritettavat rotaatiot suhteutetaan tähän tasoon. Neljästä ulottuvuudesta johtuen kvaterniot eivät

kärsi gimbal lock -ongelmasta. Neljästä ulottuvuudesta johtuen, on kuitenkin kvaternioita myös hyvin vaikeaa havainnollistaa kuvainnollisesti, mikä lisää niiden monimutkaisuutta. Kvaterniota onkin hyvin vaikea kuvitella kolmiulotteiseen tilaan intuitiivisesti, ja perusluonteeltaan kvaterniolla ei ole pakko olla fyysistä esitystä. Kolmiulotteisen liikkeen kuvaaminen neliulotteisella luvulla, ei välttämättä vaikuta intuitiiviselta. Emme kuitenkaan pysty kuvaamaan kolmiulotteista liikettä kolmella ulottuvuudella vääristämättä sitä. Kun rotaation kuvaamiseen lisätään neljäs ulottuvuus, muodostuu rotaatioiden matematiikasta lineaarista. Samalla tavalla emme pysty kuvaamaan kaksiulotteisesti maapallon karttaa vääristämättä sitä, mutta kun lisäämme kolmannen ulottuvuuden, ja luomme kartan palloksi, tulee kartasta lineaarinen. [4]

Kuvitteellisesta tilanteen *A* paikkareferenssiä mihin tahansa tilanteeseen *B* voidaan kuvata kulman $\theta$ rotaatiolla akselin $\hat{r}^A$ ympäri. Tätä rotaatiota voidaan kuvata kvaterniolla ${}^A_B\hat{q}$, joka on määritelty seuraavasti

$$ {}^A_B\hat{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta}{2} & -r_x \sin\frac{\theta}{2} & -r_y \sin\frac{\theta}{2} & -r_z \sin\frac{\theta}{2} \end{bmatrix} $$

Missä $r_x$, $r_y$, ja $r_z$ kuvaavat yksikkövektorin $\hat{r}^A$ *x*, *y*, ja *z* komponentteja tilanteessa *A*. Kvaternioiden yhdistäminen ja aritmetiikka vaativat, että kvaterniot ovat normalisoituja. Normalisoinnilla tarkoitetaan kvaternion jakamista samanlaisella kvaterniolla, jonka pituus on 1. Tästä syystä on käytännöllistä, että kaikki kvaterniot jotka kuvaavat joko paikkaa tai rotaatiota ilmaistaan yksikköpituisina.

$$ q = \|q\| $$

Kvaternion ${}^A_B\hat{q}$ konjunktiota voidaan käyttää laskemaan paikkareferenssi tilanteelle *B* tilanteesta *A*, eli toisin sanoen vaihtamaan tilanteiden paikkaa. Kvaternion ${}^A_B\hat{q}$ konjunktiota voidaan merkitä kvaterniona ${}^A_B\hat{q}^{-1}$. Samoin kvaternio ${}^A_B\hat{q}$ on kvaternion ${}^B_A\hat{q}$ konjunktio. Kvaternion konjunktio noudattaa kaavaa

$$ {}^A_B\hat{q}^{-1} = {}^B_A\hat{q} = \begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \end{bmatrix} $$

Kvaternion ${}^A_B\hat{q}$ konjunktio ${}^A_B\hat{q}^{-1}$ on siis käänteinen rotaatio kvaterniolle ${}^A_B\hat{q}$. Kvaternion konjunktion laskeminen on erittäin hyödyllistä käytännön sovellutuksia varten mm.

mikäli haluamme laskea rotaatiomuunnoksen. Mikäli kerromme kvaternion $q$ sen konjunktiolla $q^{-1}$, on tuloksena reaaliluku *R*, kuten on havainnollistettu kaavassa

$$qq^{-1} = a^2 + b^2 + c^2 + d^2 = |R|$$

Kvaternioita voidaan myös yhdistää, mikäli halutaan yhdistää kaksi eri rotaatiota. Toisin kuin normaaleita vektoreita käsiteltäessä, kun halutaan yhdistää kaksi kvaterniota, kvaternioita ei yhdistetä yhteenlaskemalla. Kun kvaterniot yhdistetään yhdeksi rotaatioksi, ne kerrotaan keskenään. Mikäli on määritelty kvaterniot ${}^{A}_{B}\hat{q}$ sekä ${}^{B}_{C}\hat{q}$, missä kvaternio ${}^{A}_{B}\hat{q}$ kuvaa rotaatiota tilanteesta *A* tilanteeseen *B*, sekä kvaternio ${}^{B}_{C}\hat{q}$ vastaavasti rotaatiota tilanteesta *B* tilanteeseen *C*, voidaan rotaatio tilanteesta *A* tilanteeseen *C* kuvata kaavalla

$${}^{A}_{C}\hat{q} = {}^{B}_{C}\hat{q} \times {}^{A}_{B}\hat{q}$$

Tulee huomioida, että kvaternioiden kertolaskut eivät ole vaihdannaisia keskenään. Tämä sääntö voidaan ilmaista matemaattisesti

$${}^{B}_{C}\hat{q} \times {}^{A}_{B}\hat{q} \neq {}^{A}_{B}\hat{q} \times {}^{B}_{C}\hat{q}$$

kahden kvaternion $q^a$ sekä $q^b$ kertolasku voidaan laskea kaavalla

$$q^a \times q^b = [q_1^a \quad q_2^a \quad q_3^a \quad q_4^a] \times [q_1^b \quad q_2^b \quad q_3^b \quad q_4^b]$$
$$= \begin{bmatrix} q_1^a q_1^b - q_2^a q_2^b - q_3^a q_3^b - q_4^a q_4^b \\ q_1^a q_2^b + q_2^a q_1^b + q_3^a q_4^b - q_4^a q_3^b \\ q_1^a q_3^b - q_2^a q_4^b + q_3^a q_1^b + q_4^a q_2^b \\ q_1^a q_4^b + q_2^a q_3^b - q_3^a q1_2^b + q_4^a q_1^b \end{bmatrix}^T$$

Kvaternioiden etuina mainittakoon mm. se, että kvaterniot ovat laskennallisesti hyvin kevyitä. Suotimet, jotka käyttävät apunaan kvaternioita, koostuvat yleisesti hyvin yksinkertaisista aritmeettisista operaatioista, eikä operaatioita ole montaa. Tämä tarkoittaa, että ne eivät rasita prosessoria samalla tavalla, kuin raskaat aritmeettiset operaatiot.

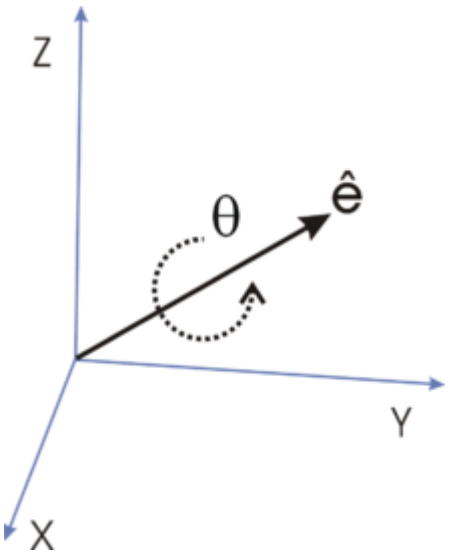
Kvaterniot kuvaavat myös rotaatioita tarkemmin sekä tehokkaammin kuin mm. Eulerin kulmat. Siinä missä Eulerin kulmat kärsivät usein rotaatioiden vääristymästä, sillä

kaikkia gimbaleita ei voida tehokkaasti kuvata pelkästään yhdellä rotaatiolla kohteesta A kohteeseen B, eivät kvaterniot kärsi samasta ongelmasta. Kvaternioden rotaatioita yhdistäessä on rotaatio kahden pisteen välillä aina suorin rotaatio näiden kahden pisteen välillä. Tämä ominaisuus juontuu kvaternioiden perimmäisestä neliulotteisesta luonteesta. Tästä syystä useat 3D-grafiikkamoottoreiden kehittäjät ovat omaksuneet kvaternioiden käytön moottoreissaan. Näihin tarjoajiin kuuluvat mm. suurimmat peliteollisuudessa käytetyt grafiikkamoottorit kuten Unity [9]. Kvaternioiden käyttö kuvatessa 3D-liikettä onkin yleisesti omaksuttu käytäntö grafiikkaesityksissä [22].

## 2.3 Vektori ja kulma

Koska tämän työn käytännön osuudessa emme käyttäneet rotaatioiden kuvaamiseen vektoria ja kulmaa, emme käy vektorin ja kulman teoriaa läpi hyvin yksityiskohtaisesti. On kuitenkin hyödyllistä huomioida kaikki mahdolliset rotaatioiden esitystavat tässä tutkielmassa, jotta voimme paremmin ymmärtää perustelut valituille esitysmuodoille.

Vektori ja kulma ovat yksi esitysmuoto rotaatiolle kolmiulotteisessa avaruudessa. Vektori ja kulma koostuva yksikkövektorista $\hat{e}$ sekä kulmasta $\theta$, joka kuvaa vektorin rotaatiota vektorin kanssa yhdensuuntaisen akselin ympäri. Tätä rakennetta pyritään kuvaamaan kuviossa 5.




Kuvio 5. Yksikkövektori sekä rotaatio yhdensuuntaisen akselin ympäri.

Rotaatiot vektorilla ja kulmalla ovat ehkä Eulerin kulmien jälkeen intuitiivisin tapa kuvata rotaatiota kolmiulotteisessa avaruudessa. Tulee kuitenkin huomioida, että mikäli rotaatioita yhdistetään käyttäen vektoria ja kulmaa, rotaatiot voivat olla harhaanjohtavia. Rotaatiot vektorilla ja kulmalla eivät ole kommutatiivisia, tarkoittaen, että rotaatioiden suoritusjärjestys vaikuttaa lopulliseen rotaatioon. Rotaatioita ei voida suoraan yhdistää käyttäen vektoria ja kulmaa. Mikäli rotaatioita halutaan yhdistää, tulee esitysmuotona käyttää joko kvaternioita tai rotaatiomatriiseja. [4]

Hyvä puoli vektorin ja kulman käytössä on se, ettei se kärsi gimbal lock -ilmiöstä. Vektorilla ja kulmalla on kuitenkin omat erikoispisteensä, joissa voidaan havaita poikkeavaa käyttäytymistä. Nämä pisteet ovat 0° sekä 180°, missä akselia ei ole määritetty.

Vektoria ja kulmaa merkitään yleisesti matematiikassa merkinnällä

$$\{ \hat{e}, \theta \} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \theta$$

Vektorin ja kulman määrittämiä rotaatioita voidaan soveltaa muille kuin yksikkövektoreille käyttäen *Rodriguesin rotaatioalgoritmia* [23]. Tämä rotaatioalgoritmi on seuraavaa muotoa

$$v_{rotaatio} = (\cos\theta)v + (\sin\theta)(\omega \times v) + (1 - \cos\theta)(\omega \cdot v)\omega$$

Missä $v$ on reaalilukujen joukkoon kuuluva vektori ja $\omega$ on rotaatioakselin suuntaa kuvaava yksikkövektori. Ilman *Rodriguesin rotaatioalgoritmia* tulee rotaatio kääntää rotaatiomatriisin muotoon, mikäli sitä halutaan soveltaa muihin kuin yksikkövektoreihin.

Vektorin ja kulman etuihin kuuluu myös se, että esitysmuotoa käyttämällä voidaan rotaatioiden muunnosnopeutta kuvata sangen helposti, sillä ne ovat suoraan verrannollisia rotaatioiden väliseen etäisyyteen.

## 2.4 Rotaatiomatriisi

Koska tämän työn käytännön osuudessa emme käyttäneet rotaatioiden kuvaamiseen rotaatiomatriisien esitysmuotoa, emme käy rotaatiomatriisien teoriaa läpi hyvin

yksityiskohtaisesti. On kuitenkin hyödyllistä huomioida kaikki mahdolliset rotaatioiden esitystavat tässä tutkielmassa, jotta voimme paremmin ymmärtää perustelut valituille esitysmuodoille.

Rotaatioita voidaan esittää myös matriiseina, jotka koostuvat yhdeksästä eri skalaariarvosta. Nämä 3x3-matriisit eivät ole kommutatiivisia sekä kärsivät erikoispisteistä kuten Eulerin kulmat. Ne kuitenkin toimivat Eulerin kulmia paremmin yhdistetään monia eri rotaatiota yhdeksi suureksi rotaatioksi. Kun kolmiulotteisessa avaruudessa pisteitä kuvataan paikkavektoreilla $(x, y, z)$ voidaan tämä vektori kertoa em. 3x3-matriisilla saadaksemme vektorin rotaation jonkin akselin suhteen.

Voimme merkitä tätä rotaatiomatriisia

$$A\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Lisäksi voimme merkitä kolmiulotteisen avaruuden vektoria edellä mainitusti

$$(x, y, z)$$

Kun kerromme vektorin $(x, y, z)$ matriisilla $A$, on tuloksena piste $(x', y', z')$, joka on piste kolmiulotteisessa avaruudessa mihin haluamme rotaation johtavan. Tätä laskutoimitusta voimme merkitä seuraavasti

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Tätä vektoria kertovaa matriisia nimitetään rotaatiomatriisiksi. Matriisin on oltava 3x3-kokoinen ortogonaalimatriisi. Ortogonaalimatriisilla tarkoitetaan, että matriisin rivit vastaavat keskenään kohtisuoria yksikkövektoreita. Ortogonaalisen matriisin determinantti on siis aina joko 1 tai -1. Kuitenkin mikäli matriisilla halutaan kuvata rotaatiota, on determinantin oltava aina positiivinen.

Rotaatiomatriiseilla on käyttäjäkuntansa varsinkin 3D-grafiikkasovelluksissa, sillä kahden eri rotaation yhdistäminen niiden avulla on suhteellisen helppoa, eikä se ole

aritmeettisesti rasittavaa. Rotaatiot yhdistetään yhdeksi halutuksi rotaatioksi yksinkertaisesti kertomalla kaksi rotaatiomatriisia keskenään. Lisäksi niiden muunnos eri esitysmuotoon on yksinkertaisempaa kuin muilla esitysmuodoilla.

## 2.5 Vertailu ja vaihdannaisuus

Kaikki edellä mainitut esitysmuodot ovat vaihdannaisia keskenään, tarkoittaen, että jäykän kappaleen asentoa kolmiulotteisessa avaruudessa kuvaavat suureet voidaan tiettyjä muunnosalgoritmeja käyttäen muuntaa toiseen esitysmuotoon. Tämä ei kuitenkaan päde kaikille rotaatioille.

Eri esitysmuotojen erikoispisteet eivät myöskään häviä kaikissa tapauksissa, mikäli rotaatioita muutetaan esitysmuodosta toiseen esim. ohjelmakoodissa.

Vaikka kaikkia eri esitysmuotoja pystytään muuntamaan toisiin esitysmuotoihin, emme käsittele tässä tutkielmassa muita kuin tutkielman kannalta oleelliset vaihdannaisalgoritmit. Nämä vaihdannaisalgoritmit ovat algoritmi Eulerin kulmista kvaternioiksi sekä algoritmi kvaternioista Eulerin kulmiksi. Tämän tutkielman käytännön osuudessa on käsitelty vain näitä em. kahta esitysmuotoa.

### 2.5.1 Eulerin kulmat kvaternioksi

Vaihdannaisalgoritmi on seuraava:

$$
\begin{aligned}
w &= c_x c_y c_z - s_x s_y s_z \\
x &= s_x s_y c_z - c_x c_y s_z \\
y &= s_x c_y c_z - c_x s_y s_z \\
z &= c_x s_y c_z - s_x c_y s_z
\end{aligned}
$$

Missä $s$ kuvaa lähtöasennon eulerin kulmien sinikerrannaisia sekä $c$ niiden kosinikerrannaisia. Toisin sanoen

$$c = \cos\frac{\theta}{2} \; sekä \; s = \sin\frac{\theta}{2}$$

$w, x, y$ ja $z$ ovat vastaavan kvaternion arvot.

### 2.5.2 Kvaterniot Eulerin kulmiksi

Kvaternioiden muuntaminen Eulerin kulmiksi voidaan suorittaa algoritmilla

$$x = \tan^{-1}(2q_y q_w - 2q_x q_z, \quad 1 - 2q_y^2 q_z^2)$$
$$y = \sin^{-1}(2q_x q_y + 2q_z q_w)$$
$$z = \tan^{-1}(2q_x q_w - 2q_y q_z, \quad 1 - 2q_x^2 q_z^2)$$

Tulee kuitenkin huomioida, että mikäli $q_x q_y + q_x q_w = \frac{1}{2}$, ei tämä muunnosalgoritmi päde, vaan tässä kyseenomaisessa erikoispisteessä pätee

$$x = 2\tan^{-1}(x, \quad w)$$
$$z = 0$$

# 3 Suotimet

## 3.1 Kalman-suodin

Kalman-suotimella pyritään suodattamaan ei-haluttua dataa mittauksista, korjaamaan häiriösignaalien aiheuttamaa virhettä sekä ennustamaan systeemin tilaa seuraavassa mittapisteessä [6]. Tyypillinen Kalman-suodin käyttää suodatukseen ja ennustamiseen kolmea eri mittapistettä systeemissä. Nämä tilat ovat systeemin tila edellisessä mittauskierroksessa, nykyisen mittauksen tila sekä ennustettu tila systeemin seuraavassa mittauskierroksessa.

Kalman-suodin käyttää edellisen mittauskierroksen tilaa sekä nykyistä tilaa ennustaakseen tilan seuraavalla mittauskierroksella, jonka jälkeen ennustettua tilaa verrataan nykyiseen mitattuun tilaan, jolloin pystytään mittamaan arvio mittausvirheestä. Koska suodin muistaa myös edellisen tilan, jota verrataan nykyiseen tilaan, ajan kuluessa mitä enemmän mittauskierroksia toteutetaan, sen parempi arvio systeemin mittausvirheestä saadaan, koska suodin korjaa omaa virhearviotaan perustuen edellisten mittausten keskiarvoihin. Tätä mittausvirheen arviota käytetään korjaamaan mittaustuloksia ja luomaan saaduista mittaustuloksista häiriöttömämpiä. [2]

### 3.1.1 Teoria ja matemaattiset perusteet

Halutulla tilalla kuvataan systeemin todellista tilaa, jota pyritään ennustamaan. Systeemin todellista tilaa ei pystytä koskaan havainnoimaan täydellisesti. Systeemin todellinen tila on aina näkymätön tarkkailijalle ja sitä pystytään kuvaamaan ainoastaan mittaustulosten $z_k$ kautta [2]. Tämän takia sitä pyritään ennustamaan mahdollisimman tarkasti ottaen huomioon systeemin eri tilat ajan hetkellä *k* sekä arvio mittausvirheestä. Tätä ilmiötä kutsutaan tilastotieteessä myös Markovin piilomalliksi.

Oletetaan, että haluttua tilaa kuvataan muuttujalla

$$x_k$$

Kalman-suodin olettaa, että haluttu tila $x_k$ pystytään aina ennustamaan systeemin edellisestä tilasta ajanhetkellä $x_{k-1}$.

Tällöin systeemin tila millä tahansa ajan hetkellä *k* voidaan merkitä seuraavasti

$$x_k = Fx_{k-1} + Bu_k + w_k$$

Missä matriisi $F$ kuvaa tilamuunnosta edellisestä tilasta $x_{k-1}$, matriisi $B$ kuvaa sisääntulomallia ja sitä sovelletaan vektoriin $u_k$, ja vektori $w_k$ kuvaa systeemin häiriötä.

$x_k$ on tilamatriisi, joka voidaan merkitä myös seuraavasti

$$x_k = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_k$$

Missä suotimen ulostulona toimii kulma $\theta$ suhteutettuna vääristymään $\dot{\theta}$. Vääristymä $\dot{\theta}$ saadaan tässä sovelluksessa käytännössä gyroskoopilta sekä kiihtyvyyssensorilta. Vääristymä kuvaa gyroskoopin kulmanopeuden virhettä. Tällöin todellinen kulmanopeus pystytään laskemaan gyroskoopin mittaustuloksen sekä vääristymän erotuksella.

Matriisia *F* sovelletaan systeemin edelliseen tilaan $x_{k-1}$. Tässä tapauksessa F voidaan määritellä

$$F = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}$$

Matriisia *F* kutsutaan yleisesti myös nimellä tilamuunnosmalli.

Muuttujaa $u_k$ kutsutaan kontrollisisääntuloksi. Tässä nimenomaisessa tapauksessa $u_k$ kuvaa kulmanopeutta, jota mitataan gyroskoopilla. Perusyhtälö voidaan tällöin ilmaista myös muodossa

$$x_k = Fx_{k-1} + B\dot{\theta} + w_k$$

Tässä yhtälössä matriisia *B* kutsutaan kontrollisisääntulomalliksi ja se on määritelty matriisina

$$B = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix}$$

Tästä yhtälöstä voimme huomata, että kertomalla kulmanopeus $\dot{\theta}$ ajan muutoksella $\Delta t$, voidaan saada suoraan arvo kulmalle $\theta$. Koska vääristymää ei lasketa suoraan perustuen kulmanopeuteen, voidaan matriisin toinen alkio olettaa nollaksi.

Viimeisenä perusyhtälön muuttujista käsitellään systeemin häiriötä kuvaava $w_k$. Kalman-suodin olettaa, että systeemin häiriö noudattaa Gaussilaista jakaumaa jonka mediaani on nolla. Tämä voidaan ilmaista yhtälönä

$$w_k \sim N(0, Q_k)$$

Missä $Q_k$ on kovarianssi *Q* ajan suhteen *k*, jota voidaan kuvata myös kovarianssimatriisina. $Q_k$ kuvaa systeemin häiriötä, ja tässä nimenomaisessa sovellutuksessa oletamme häiriön olevan kiihtyvyysanturin tilaennuste $Q_\theta$ suhteutettuna kulmanopeuden vääristymään $Q_{\dot{\theta}_b}$.

$$Q_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

Kovarianssimatriisi $Q_k$ riippuu siis ajanhetkestä *k*, jonka takia kiihtyvyysanturin vaihtelun arvio $Q_\theta$ sekä kulmanopeuden vääristymän arvio $Q_{\dot{\theta}_b}$ kerrotaan ajan

muutoksella $\Delta t$. Tämä kertominen tehdään, sillä häiriön oletetaan kasvavan ajan kasvaessa mittausten välillä. Kalman-suotimen toiminnalle on oleellista tietää sekä kiihtyvyysanturin vaihtelu $Q_\theta$ sekä kulmanopeuden vääristymän arvio $Q_{\dot{\theta}_b}$. Mikäli arvoja ei tiedetä, suodin ei toimi odotetulla tavalla. Mikäli arvioitu kulma on virheellinen tai se ajelehtii ajan kasvaessa (ns. ”driftaa”), on kulmanopeuden vääristymän arvio $Q_{\dot{\theta}_b}$ liian pieni. Jos taas arvio toimii hitaasti, on kiihtyvyysanturin vaihtelun arvio $Q_\theta$ liian suuri, jolloin suodin luottaa liikaa kulman arvioon.

Edellä mainituilla yhtälöillä voidaan kuvata systeemin todellista tilaa $x_k$, mutta pystyäksemme ennustamaan systeemin häiriötä sekä arvioimaan systeemin todellista tilaa, pitää meidän muotoilla yhtälö, jolla kuvaamme systeemin todellisen tilan sekä mittaustulosten korrelaatiota. Merkitsemme mittaustuloksia muuttujalla

$$z_k$$

Mittaustulosten $z_k$ suhdetta systeemin todelliseen tilaan $x_k$ voidaan kuvata yhtälöllä

$$z_k = Hx_k + v_k$$

Missä systeemin todellinen tila kerrotaan matriisilla H, jonka jälkeen siihen lisätään mittausvirhe $v_k$.

Matriisia H kutsutaan myös havaintomalliksi. Matriisia käytetään liittämään systeemin todellinen tila havaittuun tilaan. Havaitulla tilalla tarkoitetaan mittaustuloksiin perustuvaa tilan havainnointia, sillä systeemin todellista tilaa ei pystytä koskaan havainnoimaan. Matriisia H voidaan kuvata matriisina

$$H = [1 \quad 0]$$

Oletamme taas mittausvirheen $v_k$ noudattavan samoja lainalaisuuksia kuin systeemin häiriön $w_k$. Mittausvirheen tulee olla gaussilaisesti jakautunutta mediaanilla nolla, jossa muuttuja R kuvaa kovarianssia. Tällöin mittausvirhe $v_k$ voidaan merkitä

$$v_k \sim N(0, R)$$

Mittausvirhettä kuvaavassa kaavassa kovarianssi *R* ei kuitenkaan ole matriisi, niin kuin systeemin häiriötä kuvaava kovarianssi $Q_k$. Koska R ei ole matriisi voidaan olettaa, että mittausvirhe on yhtäläinen mittausten epätarkkuuteen, sillä yhden muuttujan kovarianssi samasta muuttujasta on sama kuin mittausvirhe. Tällöin *R* voidaan määritellä siis

$$R = E[v_k \quad v_k^T] = \ var(v_k)$$

Oletamme mittausvirheen olevan riippumaton ajasta *k*, jolloin mittausvirhe on sama kuin mittausten epätarkkuus

$$var(v_k) = var(v)$$

Mittausten epätarkkuudella $var(v)$ kuvataan suotimen luottamusta uusien mittaustulosten tarkkuuteen, siinä missä mittausvirheellä $var(v_k)$ kuvataan mittausten todellista virhettä. Mitä tarkemmin mittausten epätarkkuus $var(v)$ pystytään arvioimaan, sitä tarkemmin mittausvirheet pystytään ottamaan huomioon suotimessa. Mikäli mittausvirhe arvioidaan suotimessa liian pieneksi, mittaukset ovat luonteeltaan epätarkkoja. Mikäli mittausvirhe arvioidaan liian korkeaksi, suodin toimii hitaammin, sillä sen luottamus mittaustuloksiin on vähäistä.

Seuraavilla yhtälöillä pyrimme arvioimaan systeemin todellista tilaa $x_k$ ja sekä virhematriisia *P* millä tahansa ajan hetkellä *k*. Tätä varten on meidän määritellä muuttuja systeemin todellisen tilan arviolle. Olkoon tämä muuttuja

$$\hat{x}_k$$

Tästä muuttujasta voimme myös johtaa muuttujan systeemin edellisen tilan arviolle. Olkoon tämä

$$\hat{x}_{k-1|k-1}$$

systeemin edellisen tilan arvio perustuu edelliseen tilaan sekä sitä edeltävien tilojen arvion painotettuun keskiarvoon.

Seuraavan tilan voimme määritellä arviona tilamatriisista ajanhetkellä *k*, joka perustuu systeemin edelliseen tilaan sekä sitä edeltävien tilojen arvion painotettuun keskiarvoon. Merkittäköön tätä tilaa

$$\hat{x}_{k|k-1}$$

Tätä tilaa kutsutaan myös nimellä *priori*.
On tarpeellista määrittää vielä muuttuja tilalle, jota kutsutaan nimellä *posteriori*. Olkoon tämä muuttuja

$$\hat{x}_{k|k}$$

*Posteriori* tilalla kuvataan arviota systeemin tilasta millä tahansa ajan hetkellä *k*, kun huomioon otetaan kaikki näennäiset havainnot ennen ja mukaan lukien ajan hetken *k*.

Ensin suodin yrittää arvioida nykyistä eli *priori*-tilaa pohjaten kaikkiin edellisiin tilojen arvioihin sekä kulmanopeuden mittaukseen. Tätä voidaan kuvata yhtälöllä

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1|k-1} + B\hat{\theta}_k$$

Tämän jälkeen pyrimme selvittämään *priori*-tilan virhematriisin $P_{k|k-1}$ perustuen edellisen tilan virhematriisiin $P_{k-1|k-1}$.

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k$$

Missä *F* kuvaa tilamuunnosmallia sekä $F^T$ on tilamuunnosmallin transpoosi. Tällä matriisilla kuvaamme kuinka paljon suodin luottaa nykyisen tilan arvion nykyisiin arvoihin. Mitä pienempi matriisin arvo on, sitä enemmän luotamme arvioon nykyisestä tilasta. Koska jokaisella tilapäivityksellä virhekovariantti kasvaa, tulee matriisi kertoa tilamuunnosmallilla sekä tilamuunnosmallin transpoosilla. Tämän lisäksi yhtälöön pitää ottaa huomioon sen hetkinen virhekovarianssi $Q_k$ ajan hetkellä *k*.

Virhematriisista tässä tapauksessa muodostuu kaksi kertaa kaksi kokoinen matriisi

$$P = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}$$

*Priori* tilasta voimme johtaa yhtälön, joka kuvaa *priori*-tilan ja mittaustulosten eroa. Tätä yhtälöä kutsutaan myös nimellä innovaatio.

$$\tilde{y}_k = z_k - H\hat{x}_{k|k-1}$$

Havainnointimallia *H* käytetään mallintamaan *priori*-tilaa havainnollistettavassa tilassa, joka tässä tapauksessa on kiihtyvyysanturin mittaustulos. Näin ollen innovaatio ei ole matriisi.

$$\tilde{y}_k = [\tilde{y}]_k$$

Innovaatiolle voidaan laskea kovariantti $S_k$ kaavalla

$$S_k = HP_{k|k-1}H^T + R$$

Kovariantilla $S_k$ pyritään ennustamaan kuinka paljon luottoa suotimen tulisi antaa mittaustuloksille perustuen *priori*-tilan virhekovarianttimatriisille $P_{k|k-1}$ sekä mittauksen virhekovarianttimatriisille $R$. Havainnointimallia käytetään jälleen liittämään *priori*-tilan virhekovarianttimatriisi $P_{k|k-1}$ havainnoitavissa olevaan tilaan.

Yhtälöstä voidaan päätellä, että mittaushäiriöiden kasvaessa kovariantti $S_k$ kasvaa suhteessa. Mitä korkeamman arvon $S_k$ siis saa mittauksessa, sitä vähemmän suodin luottaa mittauksen todenperäisyyteen. $S_k$ ei myöskään ole matriisi, joten se voidaan kirjoittaa

$$S_k = [S]_k$$

Innovaatiokovariantilla pystymme laskemaan ns. Kalmanvahvistuksen $K_k$. Kalmanvahvistuksella kuvataan suotimen luottoa innovaatioon.

$$K_k = P_{k|k-1}H^T S_k^{-1}$$

Yhtälöstä pystytään päättelemään, että mikäli innovaatioon ei luoteta, niin innovaatiokovariantti $S_k$ tulee olemaan korkea, ja mikäli luotamme arvioon systeemin

tilasta, niin virhekovarianttimatriisi $P_{k|k-1}$ pienenee. Kalmanvahvistus tulee tällöin olemaan pieni. Päinvastainen pätee, mikäli luotamme innovaatioon, mutta emme arvioon systeemin tilasta.

Mikäli voidaan olettaa, että systeemin tila tiedetään ensimmäisellä mittauskierroksella, voidaan virhekovarianttimatriisi alustaa

$$P \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Tällöin Kalman -vahvistus voidaan merkitä seuraavana 2x1 matriisina

$$K = \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}$$

Laskemalla yhteen priori -tila $\hat{x}_{k|k-1}$ sekä Kalmanvahvistus $K_k$ kerrottuna innovaatiolla $\tilde{y}_k$, voidaan rakentaa yhtälö posteriori tilan arviolle.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$

Koska innovaatio $\tilde{y}_k$ on erotus mittaustulosten $z_k$ sekä arvioidun priori -tilan $\hat{x}_{k|k-1}$ välillä, voi innovaatio sekä näin ollen myös yhtälön viimeinen termi olla sekä negatiivinen, että positiivinen.

Pelkistetysti voidaan todeta, että yhtälö jatkuvasti korjaa priori-tilan $\hat{x}_{k|k-1}$ arviota, joka luotiin aikaisemmin käyttäen edellistä tilaa sekä mittaustuloksia käyttäen apunaan Kalmanvahvistusta sekä innovaatiota. Tässä nimenomaisessa tapauksessa tilaa korjataan siis gyroskoopin mittaustuloksia kiihtyvyysanturin mittaustuloksilla.

Edellä nähdystä yhtälöstä voimme vielä johtaa kaavan posteriori-tilan virhekovarianttimatriisille

$$P_{k|k} = (I - K_k H) P_{k|k-1}$$

missä $I$ on identiteettimatriisi, joka on määritelty muodossa

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Yhtälöistä voimme havaita, että yhtälö korjaa virhekovarianttimatriisia perustuen arvion korjauksen määrään.

### 3.1.2 Kalman-suotimen ohjelmointi sekä implementointi

Kalman-suodin voidaan kirjoittaa periaatteessa mille tahansa ohjelmointikielelle. Tässä luvussa käytetään usein käytettyä ns. pseudo-ohjelmointikieltä havainnollistamaan Kalman -suotimen implementoimista ohjelmointiin sekä avataan edellisen luvun teoreettiset yhtälöt. Luvussa oletetaan, että suotimelle annetut arvot ovat valmiiksi käännetty Eulerin kulmien esitysmuotoon. Kyseenomainen suotimen implementaatio ei pysty toimimaan, mikäli sisääntulona annetaan pelkkiä anturien raaka-arvoja.

Ensimmäisenä kirjoitetaan auki yhtälö priori -tilan arviosta $\hat{x}_{k|k-1}$.

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1|k-1} + B\dot{\theta}_k$$

Yhtälö voidaan kirjoittaa myös matriisimuotoon korvaamalla muuttujan edellisessä kappaleessa esitetyillä matriisiesitysmuodoilla. Tällöin voidaan muodostaa yhtälö

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k$$

Yhtälöä voidaan yksinkertaistaa toteuttamalla yhtälössä oleva matriisien kertolaskut.

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} \theta - \dot{\theta}_b \Delta t \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \dot{\theta} \Delta t \\ 0 \end{bmatrix}$$

Kaksi matriisia voidaan laskea yhteen. Tästä seuraa yhtälö

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} \theta - \dot{\theta}_b \Delta t + \dot{\theta} \Delta t \\ \dot{\theta}_b \end{bmatrix}$$

Tätä yhtälöä voidaan yksinkertaistaa edelleen ottamalla yhteiseksi jakajaksi ajan muutos $\Delta t$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} \theta - \Delta t(\dot{\theta}_b - \dot{\theta}) \\ \dot{\theta}_b \end{bmatrix}$$

Yhtälöistä nähdään, että arvio kulmasta priori -tilassa $\hat{\theta}_{k|k-1}$ on yhtä kuin arvio kulmasta edellisessä tilassa $\hat{\theta}_{k-1|k-1}$ yhteenlaskettuna kuluneeseen aikaan $\Delta t$. Koska vääristymää ei voida suoranaisesti mitata, on priori -tilan vääristymä yhtä kuin edellisen tilan vääristymä.

Pelkistetty yhtälö voidaan kirjoittaa pseudokoodina seuraavasti

```
angle = angle + (dt * rate);
Rate = newRate + Bias;
```

Seuraavaksi auki voidaan kirjoittaa priori tilan virhekovarianttimatriisin yhtälö

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k$$

Yhtälö voidaan kirjoittaa matriisimuotoon seuraavasti

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

Yhtälöä voidaan pelkistää suorittamalla matriisien kertolaskut

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \begin{bmatrix} P_{00} - \Delta t P_{10} & P_{01} - \Delta t P_{11} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t(P_{01} - \Delta t P_{11}) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

Saadut kaksi matriisia voidaan laskea yhteen

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t(P_{01} - \Delta t P_{11}) + Q_\theta \Delta t & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix}_{k-1|k-1}$$

Jonka jälkeen yhtälöä voidaan vielä pelkistää ottamalla yhteiseksi tekijäksi ajan muutos $\Delta t$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \begin{bmatrix} P_{00} - \Delta t(P_{10} - P_{01} - \Delta t P_{11} + Q_\theta) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix}$$

Tämä yhtälö voidaan kirjoittaa pseudokoodina seuraavasti

```
P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
P[0][1] -= dt * P[1][1];
P[1][0] -= dt * P[1][1];
P[1][1] += Q_gyroBias * dt;
```

Tämän jälkeen voidaan laskea innovaatio $\tilde{y}_k$ kaavasta

$$\tilde{y}_k = z_k - H\hat{x}_{k|k-1}$$

Kun kaavaan sijoitetaan matriisiarvo havaintomallille $H\hat{x}_{k|k-1}$, muodostuu yhtälöksi

$$\tilde{y}_k = z_k - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1}$$

Tämä voidaan edelleen sieventää muotoon

$$\tilde{y}_k = z_k - \theta_{k|k-1}$$

Saatu lauseke voidaan kirjoittaa pseudokoodina seuraavasti

```
y = newAngle - angle;
```

Innovaatiolle lasketaan kovariantti kaavasta

$$S_k = HP_{k|k-1}H^T + R$$

Sijoittamalla matriisimuoto havainnointimallille $H$, sekä sen transpoosille $H^T$, sekä kirjoittamalla auki matriisi priori -muodon virhekovariantille voidaan lauseke muotoilla

$$S_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + R$$

Sieventämällä edelleen voidaan lauseke kirjoittaa muotoon

$$S_k = P_{00}{}_{k|k-1} + R$$

Sijoittamalla mittauksen virhekovariantti $R$, saadaan lausekkeen lopullinen muoto

$$S_k = P_{00}{}_{k|k-1} + var(v)$$

Huomionarvoista on, että kyseisessä tapauksessa innovaatiosta ei muodostu matriisia. Innovaatio voi kuitenkin olla luonteeltaan myös matriisi, erilaisissa tapauksissa, joissa mm. havainnointimallin arvot ovat erilaiset. Saatu lopullinen yhtälö voidaan kirjoittaa ohjelmakoodina seuraavalla lausekkeella

```
S = P[0][0] + R_angle;
```

Kuten edellä on mainittu, pystytään innovaatiokovariantin perusteella laskemaan Kalmanvahvistus algoritmille. Kalmanvahvistus noudattaa kaavaa

$$K_k = P_{k|k-1} H^T S_k^{-1}$$

Joka voidaan kirjoittaa auki sijoittamalla arvot priori-tilan virhekovarianttimatriisille sekä havainnointimallille. Edellä totesimme myös itse Kalmanvahvistuksen olevan luonteeltaan matriisi.

$$K_k = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} S_k^{-1}$$

Sijoitetaan lausekkeeseen näkyviin myös Kalmanvahvistuksen olennainen luonne matriisina sekä sievennetään yhtälön oikean puolen matriisit kertomalla ne yhteen.

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1} S_k^{-1}$$

Tässä nimenomaisessa tapauksessa, missä innovaation kovariantti $S_k$ ei ole luonteeltaan matriisi, voidaan yhtälö sieventää muotoon

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \frac{\begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1}}{S_k}$$

Tulee huomioida, että innovaation kovariantti $S_k$ voi muissa tapauksissa saada myös matriisimuodon. Tällöin ei voida yksinkertaisesti jakaa vihekovarianttimatriisia $P$ matriisilla $S_k$, vaan tulee laskea matriisille $S_k$ sen käänteismatriisi $S_k^{-1}$.

Tämä sievennetty muoto voidaan ilmaista ohjelmakoodissa kahdella lausekkeella

```
K[0] = P[0][0] / S;
K[1] = P[1][0] / S;
```

Seuraavaksi voidaan kirjoittaa yhtälö posteriori -tilan arviolle $\hat{x}_{k|k}$. Posteriori -tilan yhtälö noudattaa muotoa

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$

Sijoittamalla tunnetut arvot Kalmanvahvistukselle $K_k$ sekä innovaation arviolle $\tilde{y}_k$, kertomalla ne keskenään sekä korvaamalla posteriori- ja priori-tilojen arviot niiden matriisimuodoilla, voidaan yhtälö ilmaista myös muodossa

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k} = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \tilde{y} \\ K_1 \tilde{y} \end{bmatrix}_k$$

Sama yhtälö voidaan kirjoittaa jälleen ohjelmakoodina seuraavasti

```
angle = K[0] * y;
bias = K[1] * y;
```

Viimeisenä tulee laskea arvo virhekovarianttimatriisille $P_{k|k}$. Virhekovarianttimatriisi $P_{k|k}$ on kaksi kertaa kaksi matriisi, jonka alkioiden laskemiseen voidaan noudattaa kaavaa

$$P_{k|k} = (I - K_k H) P_{k|k-1}$$

Missä aikaisemmin olemme todenneet $I$:n olevan ennalta määritelty identiteettimatriisi. Muut lausekkeen muuttujat ovat tunnettuja. Sijoittamalla matriisimuodot tunnettuihin muuttujiin, saa yhtälö muodon

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = (\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 \\ K_1 \end{bmatrix} [1 \quad 0]) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1}$$

Yhtälö voidaan sieventää toteuttamalla yhtälön oikealla puolella sijaitsevat aritmeettiset laskutoimitukset

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = (\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 & 0 \\ K_1 & 0 \end{bmatrix}) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1}$$

Joka edelleen sievenee muotoon

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} - \begin{bmatrix} K_0 P_{00} & K_0 P_{01} \\ K_1 P_{00} & K_1 P_{01} \end{bmatrix}$$

Tämä yhtälö voidaan kirjoittaa edelleen ohjelmakoodiin muodossa

```
P[0][0] -= K[0] * P[0][0];
P[0][1] -= K[0] * P[0][1];
P[1][0] -= K[1] * P[0][0];
P[1][1] -= K[1] * P[0][1];
```

Huomautettava on, että on tärkeää määrittää ohjelmakoodissa arvot halutulle tilalle, mikäli suotimen ulostuloarvoa halutaan käyttää heti käynnistyksessä. Mikäli arvoja ei määritä, ei suodin pysty laskemaan arvoja, ennen kuin määrätty arvo suotimen läpikäyntejä on suoritettu. On myös huomattava, että kyseisessä Kalman-suotimen implementoinnissa ei ole otettu huomioon Gimbal lock -tilanteita. Suodin ei pysty luonnollisesti myöskään käsittelemään näitä tiloja, sillä se antaa tulosarvoinaan Eulerin kulmia. Tämä tarkoittaa, ettei implementoitua Kalman-suodinta pystytä käyttämään, mikäli on odotettavissa, että tapahtuu rotaatioita täydessä 3D-tilassa. Nämä

erikoisarvot voidaan kuitenkin ottaa huomioon yksittäistapauksina, mikäli kyseinen toiminnallisuus halutaan kalman suotimeen implementoida.

### 3.1.3 Kalman-suodin käytännössä

Kalman-suotimen ehdottomiin etuihin kuuluu sen stabiilius. Vain harva suodin pystyy poistamaan mittaustuloksista epätäsmällisyyksiä sekä kohinaa yhtä tehokkaasti kuin Kalman-suodin.

Kalman-suotimen haittoihin kuuluu kolmiulotteisessa tilapaikannuksessa sen pohjautuminen Eulerin kulmiin. Mikäli systeemissä halutaan ottaa huomioon erikoispisteet, tulee systeemiin joko lisätä siis lisää akseleita, tai muuntaa suotimelta saadut mittaustulokset toisenlaiseen esitysmuotoon.

Tämän lisäksi Kalman-suodin, ja erityisesti ns. laajennettu Kalman-suodin on laskennallisesti erittäin raskas, ja vaatii optimaaliseen suoritukseen suhteellisen nopeita kellotaajuuksia. Kalman-suodin vaatii monta iteraatiota tuottaakseen käyttökelpoista dataa, mikä pakottaa suorituksen kellotaajuuden olemaan korkea.

### 3.1.4 Käyttökohteet

Kalman-suodin on haitoistaan huolimatta erittäin käytetty sekä suosittu suodin teollisuuden parissa. Kalman-suodinta löytyykin implementoituna erityisesti mm. Ilmailu- sekä avaruusalan, robotiikan, navigaation sekä muun liikenteen alalta, mutta sillä on käyttökohteita myös kaikessa muussa mittausteknologiassa, missä mittaustulosten stabiiliudella on tärkeä osa mittausta. Kalman-suotimesta sekä ns. laajennetusta Kalman-suotimesta onkin muodostunut pitkälti standardi teollisuudessa.

Erityisen suuri rooli Kalman-suotimella on ollut avaruus- sekä ilmailuteknologian kehityksessä. Kalman-suodinta on käytetty lukuisilla avaruuslennoilla 1960-luvulta lähtien. Tämän lisäksi nykyinen GPS-tekniikka pohjautuu suurelta osin Kalman-suotimen suomaan kohinan poistoon. Kalman-suodin sopii käytettävyydeltään erityisen hyvin systeemeihin, joiden kellotaajuus sekä laskentateho ovat riittäviä tukemaan suotimen vaatimaa nopeaa suoritustaajuutta sekä eivät kärsi vaadittavasta suuresta laskentatehosta. Tämän lisäksi suodin sopii systeemeihin, joissa mittaustulosten

tarkkuudella sekä stabiiliudella on kriittinen rooli tai systeemeihin, joissa esiintyy suuria määriä kohinaa.

## 3.2 Madgwickin suodin

Madgwickin suodin on Sebastian Madgwickin vuonna 2009 kehittämä suodinalgoritmi käytettäväksi IMU- sekä MARG-järjestelmiin [1]. Suodin perustuu *Robert Mahonyn* kehittämään komplementaarisuotimeen, kuitenkin yhdistäen komplementaarisuotimen toiminnallisuuden kvaterniopohjaiseen esitykseen. Kvaterniopohjaisuus eliminoi Eulerin kulmiin liittyvät erikoispisteet sekä ongelmat gimbal lockin kanssa. Tämän lisäksi kvaterniopohjaisuus tarkoittaa, että suodin on aritmeettisilta operaatioiltaan erittäin kevyt, rasittaen prosessoria erittäin vähän. Madgwick toteaa omassa tutkielmassaan suotimessa olevan maksimissaan 227 suoritettavaa aritmeettista operaatiota [1]. Suotimen etuihin kuuluu myös, että se on toiminnallinen myös alhaisilla taajuuksilla, siinä, missä mm. Kalman-suodin vaatii tehokkaaseen toimintaan korkean kellotaajuuden suotimen ajamiseksi.

### 3.2.1 Teoria

Madgwick -suodin on suunniteltu käytettäväksi erityisesti tilapaikannussysteemeissä, jotka koostuvat joko kuudesta tai yhdeksästä vapausakselista [1]. Tästä syystä johtuen on käytännöllistä selittää suotimen teoria pohjautuen oletukseen, että meillä on käytössämme vähintään 3-akselinen gyroskooppi sekä 3-akselinen kiihtyvyysanturi.

Kolmiakselinen gyroskooppi mittaa kulmanopeutta akseleiden *x*, *y* ja *z* suhteen. Voimme kutsua näitä kulmanopeuksia vastaavasti nimillä $\omega_x$, $\omega_y$ ja $\omega_z$. Mikäli luomme näistä kulmanopeuksista vektorin $\bar{s}_\omega$, noudattaa vektori kaavaa

$$\bar{s}_\omega = [0 \quad \omega_x \quad \omega_y \quad \omega_z]$$

Tällöin voimme luoda vektorista $\bar{s}_\omega$ sekä maapallon tilaa kuvaavasta kvaterniosta ${}^S_E\hat{q}$ kvaternioesityksen ilmaistuna kvaternion derivaatalla ${}^S_E\dot{q}$, joka kuvaa muutoksen nopeutta sensorin tilan ja maapallon tilan välillä, Tämä voidaan ilmaista kaavalla

$${}^S_E\dot{q} = \frac{1}{2} {}^S_E\hat{q} \times \bar{s}_\omega$$

Maapallon tilaa suhteessa sensorin tilaan millä tahansa ajan hetkellä $t$ voidaan kuvata integroimalla sensorin tilaa kuvaava kvaternion derivaatta ${}^{S}_{E}\dot{q}_{\omega t}$ kaavaan. Tällöin saamme tulokseksi kaavan

$$ {}^{S}_{E}\dot{q}_{\omega t} = \frac{1}{2} {}^{S}_{E}\hat{q}_{0t-1} \times \bar{s}_{\omega t} $$

Missä $\frac{1}{2} {}^{S}_{E}\hat{q}_{0t-1}$ kuvaa kvaterniota, joka on arvio edellisestä tilasta, $\bar{s}_{\omega t}$ kuvaa mitattua kulmanopeutta ajan hetkellä $t$. Tästä voimme edelleen johtaa kaavan kvaterniolle ${}^{S}_{E}q_{\omega t}$ kuvaten sillä kvaterniota todelliselle kulmanopeudelle millä tahansa ajan hetkellä $t$.

$$ {}^{S}_{E}q_{\omega t} = {}^{S}_{E}\hat{q}_{0t-1} + {}^{S}_{E}\dot{q}_{\omega t}\Delta t $$

Missä $\Delta t$ kuvaa näytteenottotaajuutta. Tämä kaava olettaa, että alustavat orientaatio maapallon suhteen tiedetään.

Kolmeakselinen kiihtyvyysanturi mittaa anturiin kohdistuvan painovoimakentän suuntaa sekä suuruutta yhdistettynä lineaarisiin kiihtyvyyksiin, jotka aiheutuvat anturin liikkeestä sekä siihen kohdistetuista voimista, jotka aiheuttavat kiihtyvyyksiä. Mikäli oletetaan, että systeemissä sijaitsee kolmeakselinen elektroninen magneettianturi, mittaa magneettianturi maapallon magneettikentän suuntaa sekä suuruutta, ottaen huomioon paikalliset vääristymät magneettikentässä.

Mikäli maan magneetti- tai painovoimakentän suunta tiedetään senhetkisessä maapallon tilassa, pystytään määrittelemään anturin tila suhteessa maapallon tilaan tarkastelemalla senhetkisiä maapallon magneetti- sekä painovoimakenttiä. Pitää kuitenkin ottaa huomioon, että ei ole olemassa pelkästään yhtä esitysmuotoa millekään yhdelle mittaustuloksista määritetylle orientaatiolle. Mikä tahansa yksi orientaatio pystytään esittämään äärettömällä määrällä esitysmuotojen yhdistelmiä. Kvaternioesitys vaatii toimiakseen esitysmuodon, missä yksikään orientaatioista suhteessa voimia kuvaavaan kenttään ei ole tuntematon. Tämä ongelma voidaan ehkäistä määrittelemällä algoritmi, jossa anturin orientaatiota kvaterniovektorina ${}^{S}_{E}\hat{q}$ pidetään ennalta määrätyn referenssivektorin suuntaisena vallitsevaan painovoimakenttään $\hat{d}_E$ nähden.

$$\min_{{}^{S}_{E}\hat{q}\in R} f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right)$$

Missä yhtälöä voidaan kuvata

$$f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right) = {}^{S}_{E}\hat{q}^{-1} \times \hat{d}_E \times {}^{S}_{E}\hat{q} - \hat{s}_S$$

Missä ${}^{S}_{E}\hat{q}$, $\hat{d}_E$ sekä $\hat{s}_S$ ovat kvaterniovektoreita, jotka määritellään seuraavasti

$${}^{S}_{E}\hat{q} = [q_1 \quad q_2 \quad q_3 \quad q_4]$$

$$\hat{d}_E = [0 \quad d_x \quad d_y \quad d_z]$$

$$\hat{s}_S = [0 \quad s_x \quad s_y \quad s_z]$$

Tämän lisäksi Johan Sebastian Madgwick on päättänyt käyttää filtterissään optimisaatioalgoritmina viettävää gradienttia (ks. englannin ”Gradient descent”), jolla pyritään määrittelemään paikallinen minimi funktiolausekkeelle. Optimisaatioalgoritmia perustellaan sen helppokäyttöisyydellä niin implementaation kuin laskennallisuuden suhteen.

Viettävä gradientti *n* määrälle iteraatioita voidaan määritellä Madgwickin suotimessa

$${}^{S}_{E}q_{k+1} = {}^{S}_{E}q_k - \mu \frac{\nabla f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right)}{\left\|\nabla f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right)\right\|}, missä\ k = 0,1,2,\dots,n$$

Missä ${}^{S}_{E}q_{k+1}$ kuvaa arviota oikeasta orientaatiosta, pohjautuen edelliseen arvioon orientaatiosta ${}^{S}_{E}q_k$ sekä muutoksen mitattuun suuruuteen $\mu$.

Tälle yhtälölle voimme muodostaa gradientin edellä määritetyn anturin kvaterniovektorin ${}^{S}_{E}\hat{q}$ avulla.

$$\nabla f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right) = J^T\left({}^{S}_{E}\hat{q}, \hat{d}_E\right) f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right)$$

Missä $f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right)$ voidaan määritellä vektorimatriisina

$$f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right) = \begin{bmatrix} 2d_x\left(\frac{1}{2} - q_3^2 - q_4^2\right) + 2d_y(q_1q_4 + q_2q_3) + 2d_z(q_2q_4 - q_1q_3) - s_x \\ 2d_x(q_2q_3 - q_1q_4) + 2d_y(\frac{1}{2} - q_2^2 - q_4^2) + 2d_z(q_1q_2 - q_3q_4) - s_y \\ 2d_x(q_1q_3 - q_2q_4) + 2d_y(q_3q_4 + q_1q_2) + 2d_z\left(\frac{1}{2} - q_2^2 - q_3^2\right) - s_z \end{bmatrix}$$

Samalla tavalla $J^T\left({}^{S}_{E}\hat{q}, \hat{d}_E\right)$ voidaan määritellä vektorimatriisina

$$J^T\left({}^{S}_{E}\hat{q}, \hat{d}_E\right) = \begin{bmatrix} 2d_yq_4 - 2d_zq_3 & 2d_yq_3 - 2d_zq_4 & -4d_xq_3 + 2d_yq_2 - 2d_zq_1 & -4d_xq_4 + 2d_yq_1 - 2d_zq_2 \\ -2d_xq_4 + 2d_zq_2 & 2d_xq_3 - 4d_yq_2 + 2d_zq_1 & 2d_xq_2 + 2d_zq_4 & -2d_xq_1 - 4d_yq_4 + 2d_zq_3 \\ 2d_xq_3 - 2d_yq_2 & 2d_xq_4 - 2d_yq_1 - 4d_zq_2 & 2d_xq_1 + 2d_yq_4 - 4d_zq_3 & 2d_xq_2 + 2d_yq_3 \end{bmatrix}$$

Edellä mainitut lausekkeet kuvaavat tilannetta, jossa algoritmia voidaan soveltaa mihin tahansa tilanteeseen missä voimaa aiheuttavan kentän suunta voidaan määritellä vapaasti. Mikäli kuitenkin oletamme, että painovoima on aina samansuuntainen koordinaatiston *z*-akselin kanssa, voimme korvata muuttujan $\hat{s}_S$ normalisoidulla kiihtyvyysanturin mittauksella $\hat{a}_S$, joka voidaan määritellä

$$\hat{a}_S = [0 \quad a_x \quad a_y \quad a_z]$$

Sekä korvata muuttujan $\hat{d}_E$ vastaamaan vektoria, joka on samansuuntainen painovoiman komponentin $\hat{g}_E$ kanssa, jossa $\hat{g}_E$ määritellään

$$\hat{g}_E = [0 \quad 0 \quad 0 \quad 1]$$

Tällöin vektorimatriisit sieventyvät muotoon

$$f\left({}^{S}_{E}\hat{q}, \hat{d}_E, \hat{s}_S\right) = \begin{bmatrix} 2(q_2q_4 - q_1q_3) - a_x \\ 2(q_1q_2 - q_3q_4) - a_y \\ 2\left(\frac{1}{2} - q_2^2 - q_3^2\right) - a_z \end{bmatrix}$$

sekä vastaavasti

$$J^T\left({}^{S}_{E}\hat{q}, \hat{d}_E\right) = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix}$$

Vastaavat lausekkeet voidaan määritellä laskemaan vastaavat suureet magneettikentälle. Käytettäessä normalisoituja mittaustuloksia magneettianturilta, jolloin korvaamme painovoimakentän $\hat{d}_E$ magneettikentällä $\hat{b}_E$ sekä normalisoidun mitatun kiihtyvyyskvaternion $\hat{a}_S$ magneettisensorin mittauksella $\widehat{m}_S$. Vastaavat arvot suureille, mikäli oletamme taas mittausarvojen olevan normalisoituja, ovat siis

$$\widehat{m}_S = [0 \quad m_x \quad m_y \quad m_z]$$

sekä

$$\hat{b}_E = [0 \quad b_x \quad 0 \quad b_z]$$

Emme kuitenkaan voi päätellä sensorin todellista orientaatiota, yksittäisistä mittauksista magneettikentälle tai painovoimakentälle, vaan sekä mittaustulokset sensorilta, että vastaavat referenssivektorit tulee yhdistää. Tämä voidaan toteuttaa kuvaamalla kentät yhdessä matriisissa noudattaen edellä mainittua kvaternioesitystä $f\left({}^S_E\hat{q}, \hat{d}_E, \hat{s}_S\right)$ olettaen, että missä tahansa tilanteessa magneettikentän komponentti $b_x \neq 0$.

$$f_{gb}\left({}^S_E\hat{q}, \hat{a}_S, \hat{b}_E, \widehat{m}_S\right) = \begin{bmatrix} f_{gb}\left({}^S_E\hat{q}, \hat{a}_S\right) \\ f_{gb}\left({}^S_E\hat{q}, \hat{b}_E, \widehat{m}_S\right) \end{bmatrix}$$

sekä vastaavasti referenssipinta lausekkeella

$$J_{gb}\left({}^S_E\hat{q}, \hat{b}_E\right) = \begin{bmatrix} J_g^T\left({}^S_E\hat{q}\right) \\ J_b^T\left({}^S_E\hat{q}, \hat{b}_E\right) \end{bmatrix}$$

Arvioitu todellinen orientaatio ${}^S_E q_t$ millä tahansa ajan hetkellä *t* voidaan laskea kaavalla

$${}^S_E q_t = {}^S_E\hat{q}_{t-1} - \mu_t \frac{\nabla f}{\|\Delta f\|}$$

Missä ${}^S_E\hat{q}_{t-1}$ on edellinen arvioitu todellinen orientaatio sekä $\mu_t$ on mittauksen muutos ajanhetkellä *t*. Gradientti $\nabla f$ pohjautuu antureilta mitattuihin arvoihin $\hat{a}_{St}$ sekä $\widehat{m}_{St}$, jotka kuvaavat mitattua kiihtyvyys- sekä magneettikenttää ajan hetkellä *t*. Gradientti voidaan laskea kaavalla

$$\nabla f = \begin{cases} J_g^T({}_E^S\hat{q}_{t-1}) f_g({}_E^S\hat{q}_{t-1}, \hat{a}_S) \\ J_{gb}^T({}_E^S\hat{q}_{t-1}, \hat{b}_E\,) f_{gb}({}_E^S\hat{q}_{t-1}, \hat{a}_S, \hat{b}_E, \hat{m}_S) \end{cases}$$

Mittauksen muutos $\mu_t$ ajan hetkellä *t* tulisi valita siten, että se on yhtä suuri tai suurempi kuin fyysinen orientaation muutos. Mittauksen muutoksen optimaalinen arvo tulee siis määritellä rajoittamalla arvioitua todellista orientaatiota ${}_E^S q_t$ niin, ettei se koskaan ylitä todellista fyysistä orientaation muutosta. Tästä voimme johtaa kaavan mittauksen muutoksen laskemiseksi.

$$\mu_t = \alpha \left| {}_E^S q_{\omega t} \right| \Delta t$$

Missä ${}_E^S q_{\omega t}$ on edellä mainittu gyroskoopin mittaamaa muutosnopeus ja $\Delta t$ on mittausperiodi.

Koska Madgwickin suodin käyttää sisääntulonaan raakadataa sensoreilta, on suotimessa otettava erikseen huomioon gyroskoopin arvon ajelehtiminen ajan myötä. Normalisoidun mittaustuloksen virheen arvio ${}_S^E\hat{q}_{te}$ voidaan ilmaista jokaisen gyroskoopin kolmen akselin virheenä kaavalla

$$\omega_{te}^S = 2\,{}_E^S q_{t-1}^{-1} \times {}_S^E\hat{q}_{te}$$

Missä $\omega_{te}^S$ on arvioitu gyroskoopin virhe ajan hetkellä *t*. Todellinen gyroskoopin kumulatiivinen mittausvirhe $\omega_{bt}^S$ ajan myötä voidaan ilmaista mittausvirheen arvion sarjan summan mediaanilla.

$$\omega_{bt}^S = \sum_t \Delta t\, \omega_{te}^S$$

Kompensoitu gyroskoopin mittaustulos $\omega_{ct}^S$ ajan hetkellä *t* saadaan erottamalla kumulatiivinen gyroskoopin mittausvirhe $\omega_{bt}^S$ gyroskoopin mittaustuloksesta $\omega_t^S$.

$$\omega_{ct}^S = \omega_t^S - \omega_{bt}^S$$

Kompensoitua gyroskoopin mittaustulosta $\omega_{ct}^S$ voidaan käyttää yllä mainituissa lausekkeissa suoraan gyroskoopin mittaustulosten $\omega^S$ sijalla. Tällöin mittausvirheen suuruus jokaisessa gyroskoopin akselissa on yksikkökvaternion suuruinen. Tällöin

integraaliulostulo suotimelle määrittää suoraan arvioidun gyroskoopin virheen muutosnopeuden.

### 3.2.2 Madgwickin suodin käytännössä

Madgwickin suotimen ehdottomiin etuihin kuuluu sen kvaterniopohjaisuus. Siinä perinteisemmät suotimet kärsivät joko Eulerin kulmien erikoispisteiden ongelmista tai rotaatiomatriisien tuomasta rajallisuudesta, eivät kvaterniot kärsi näistä ongelmista. Kvaterniot eivät neliulotteisuutensa takia kärsi Gimbal lockista ja niillä pystytään kuvaamaan tehokkaasti niin staattisia kappaleita sekä rotaatioita. Kvaternioiden etuja on käsitelty perusteellisemmin luvussa 2.2.

Madgwickin suodin koostuu hyvin pienestä määrästä aritmeettisia operaatioita. Tästä syystä se on prosessorille erittäin kevyt suodin. Toinen prosessointiin liittyvä etu Madgwickin suotimessa on, että se ei ole riippuvainen suotimen suoritusnopeudesta. Tällaisena sitä pystytäänkin ajamaan suhteellisen vaivatta myös prosessoreilla, joiden kellotaajuus on pieni, tai mikäli jostain muusta syystä on suodin ajastettava pienelle kellotaajuudelle.

Etuihin Madgwickin suotimessa kuuluu myös se, että suodin käsittelee puhdasta raakadataa antureilta. Näin ollen suotimelle annettavaa dataa ei täydy esikäsitellä mitenkään, siinä missä mm. Kalman-suotimelle annettava data on aina esikäsiteltävä muotoon, jossa se halutaan esittää. Koska dataa ei täydy esikäsitellä, pienentää se algoritmin implementointiin vaadittavaa muistia.

Madgwickin suotimen haittoihin kuuluvat sen spesifisyys sekä kompleksisuus. Suodin on suunniteltu pelkästään IMU- sekä MARG-laitteiden suodatukseen, eikä sitä pystytä soveltamaan niin suurella skaalalla kuin muita teollisuuden filttereitä. Sen lisäksi suotimen kustomoiminen on vaikeaa, sillä suodin nojaa vahvasti kompleksiin matematiikkaan ja edellä mainittuihin kvaternioihin. Mikäli käyttökohde ei pysty käsittelemään kvaternioita, ei Madgwickin suodinta pystytä implementoimaan.

Toinen haittapuoli on se, että käsitelläkseen kvaternioita, on suorittimen kyettävä käsittelemään liukulukuja. Sulautettuihin järjestelmiin suodinta implementoidessa on tämä otettava erityisesti huomioon, sillä liukulukujen käsittely ei sulautetuissa

suorittimissa ole itsestäänselvyys. Liukuluvut kuluttavat myöskin suuren määrän ohjelmamuistia erityisesti sulautetuissa järjestelmissä.

### 3.2.3 Käyttökohteet

Madgwickin suotimen käyttökohteet ovat hyvin rajatut. Itse suodin on suunniteltu alun perin toimimaan pelkästään MARG-systeemeille. Suodin olettaa käytettävissä olevan kolmiakselinen gyroskooppi, kolmiakselinen kiihtyvyysanturi sekä kolmiakselinen magneettianturi. Tämä rajaa Madgwickin suotimen käyttökohteet pelkkään inertianavigointiin sekä kappaleen asennon tunnistukseen. Myöhemmin Johan Sebastian Madgwick on kuitenkin kehittänyt sekä julkaissut myös version suotimesta, joka toimii ilman magneettianturia, näin ollen laajentavan sen käytön myös IMU-järjestelmiin.

Käytännön käyttökohteiksi Madgwickin suotimelle kuuluvat siis kaikki sovellutukset, joissa asennon määrittäminen on tärkeää, eikä prosessointiin ole saatavilla yletöntä määrää resursseja. Näihin käyttökohteisiin lukeutuvat mm. ilmailuala sekä yleinen navigaatio. Muun muassa lennokkiharrastelijat ovat omaksuneet suotimen käytön viime vuosina olennaiseksi osaksi järjestelmäarkkitehtuuriaan.

Vaikka edellä on mainittu puhtaan Madgwickin suotimen käytön olevan rajattu IMU- sekä MARG-laitteisiin, on markkinoille ilmestynyt myös kolmansien osapuolien kehittämiä suotimia muihin käyttökohteisiin, jotka pohjaavat toimintansa Madgwickin suotimeen. Näihin kohteisiin lukeutuvat kaikki sovellutukset, jotka käyttävät hyväkseen kvaternioita. Muun muassa monelle pelialan grafiikkamoottorille on kehitetty avoimen lähdekoodin sovellutuksia Madgwickin suotimesta laajentaen suotimen käyttökohteita 3D-grafiikkamoottoreihin.

## 3.3 Komplementaarisuodin

Vaikka tämän insinöörityön käytännön osuudessa ei ole käytetty varsinaisesti komplementaarisuodinta, on tärkeää ymmärtää komplementaarisuotimen perusteet, sillä käytetty Madgwickin suodin pohjaa aritmetiikkansa juuri komplementaarisuotimeen.

Komplementaarisuodin on *Robert Mahonyn* kehittämä suodinalgoritmi, joka yhdistää anturidataa sekä kolmiakselisesta gyroskoopista, että kolmiakselisesta kiihtyvyysanturista. Suodin on kehitetty, jotta pystyttäisiin yhdistämään Kalman-suotimen suoma sensorifuusio suotimeen, joka olisi aritmeettisesti tehokas. Näin suodin olisi laskennallisuudeltaan erittäin tehokas, mutta kuitenkin tuoden sensorifuusion käytännöllisyyden algoritmiin. [10]

### 3.3.1 Teoria

Komplementaarisuodin yhdistää antureilta saadun tiedon yhdessä iteraatiossa. Tämä tarkoittaa, ettei se vaadi Kalman-suotimen tavoin erillistä aikaa alustukseen. Suodin yhdistää tietoa kolmiakseliselta gyroskoopilta sekä kolmiakseliselta kiihtyvyysanturilta. Periaatteessa suodin koostuu yhdestä low-pass suotimesta sekä yhdestä high-pass suotimesta sekä algoritmista jolla suotimien läpi ajettu data yhdistetään käyttämällä eri antureille määritettyjä luotettavuusarvoja [11]. Tällä tavoin luotettavaa kulma-arvoa voidaan lukea käyttämällä gyroskooppia, mutta kuitenkin voidaan korjata gyroskoopin arvon luontainen ajelehtiminen käyttäen kiihtyvyysanturilta saatuja arvoja.

Jos merkitsemme kiihtyvyysanturilta saatua arvoa muuttujalla $\theta_a$ sekä gyroskoopilta saatua arvoa muuttujalla $\theta_g$. Merkitsemme gyroskoopilta saadun arvon integraalia muuttujalla $\dot{\theta}_g$. Integraali otetaan muuttaaksemme gyroskoopin arvon kulmanopeudesta kulmaksi.

$$\dot{\theta}_g = \theta_g * \Delta t$$

Missä $\Delta t$ on ajan muutos.

Oletamme arvojen olevan jo ajettuja niille määritetyn elektronisen suotimen läpi. Gyroskoopilta saadun arvon sekä kiihtyvyysanturilta saadun arvon summan tulee olla 1, jotta luotettavuuskerroin on mahdollista määrittää [10]. Voimme merkitä tätä seuraavasti

$$\dot{\theta}_g{}^{\varepsilon} + \theta_a{}^{\varepsilon} = 1$$

Missä $\dot{\theta}_g{}^{\varepsilon}$ on luotettavuuskerroin gyroskoopin arvolle sekä $\theta_a{}^{\varepsilon}$ luotettavuuskerroin kiihtyvyysanturin antamalle arvolle. Luotettavuuskertoimen arvot tulee määrittää jokaiselle sovellukselle erikseen, kuitenkin niin, että ne noudattavat em. sääntöä.

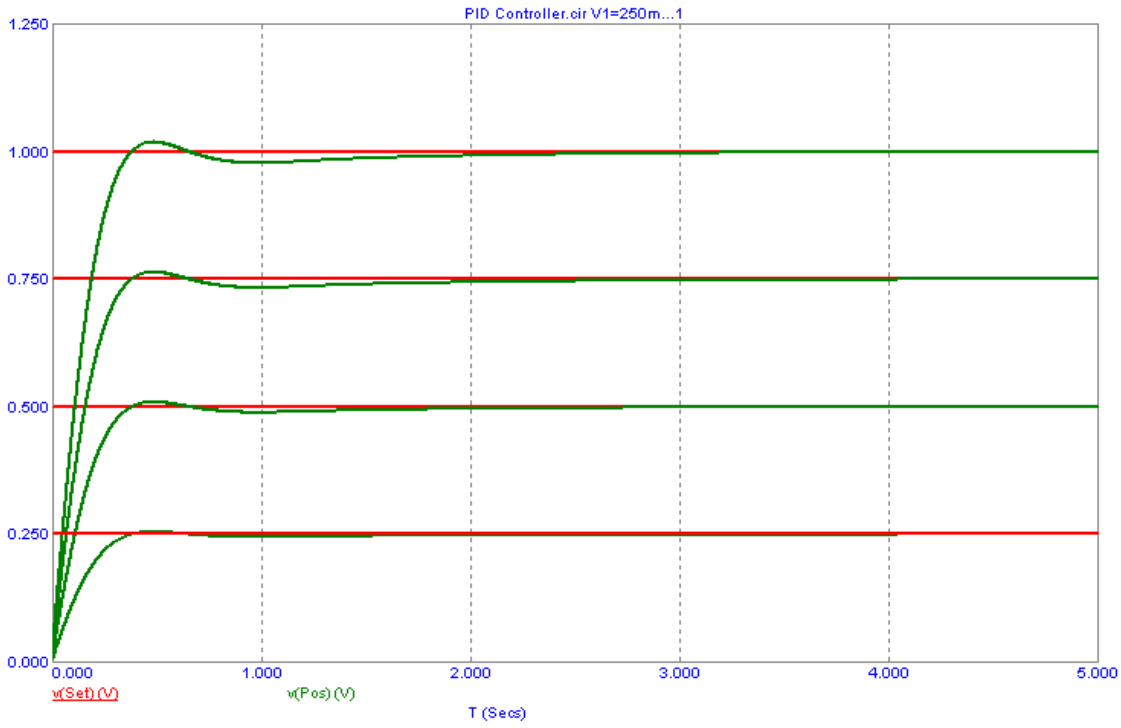
Koska haluamme kuitenkin yhdistää todellisen kulman ajanhetkellä $t$ edellisen iteraation kulma-arvoon, muotoilemme lopullisen lausekkeen muotoon

$$\alpha = \dot{\theta}_g{}^{\varepsilon}\left(\dot{\theta}_g * \alpha_{t-1}\right) + \theta_a{}^{\varepsilon}\theta_a$$

Missä $\alpha$ on kulma ajanhetkellä $t$ ja $\alpha_{t-1}$ on kulma suotimen edellisessä iteraatiossa.

## 3.4 PID-säädin

PID -säädin säätöteknillinen yleissäädin. PID -säädin on yksi yleiskäyttöisimmistä sekä myös käytetyimmistä säätimistä teollisuudessa [15 & 16]. Sen algoritmi koostuu kolmesta termistä jotka ovat Integraali-, derivaatta- sekä summatermi. Näiden termien englanninkieliset nimet (*proportional, integral, derivative*) muodostavat säätimen nimityksen, PID. Säätimen tarkoitus on yleisimmin ”pehmentää” signaalia sen arvon muuttuessa. Signaalin tulee saavuttaa haluttu arvo mahdollisimman nopeassa ajassa, kuitenkin niin, ettei signaaliin muodostu ylimääräisiä piikkejä tai kaikuja. PID säätimen vaikutusta signaaliin on kuvattu kuviossa 6.



Kuvio 6. PID -säätimen vaikutus signaaliin sen vaihtaessa arvoa.

### 3.4.1 Teoria

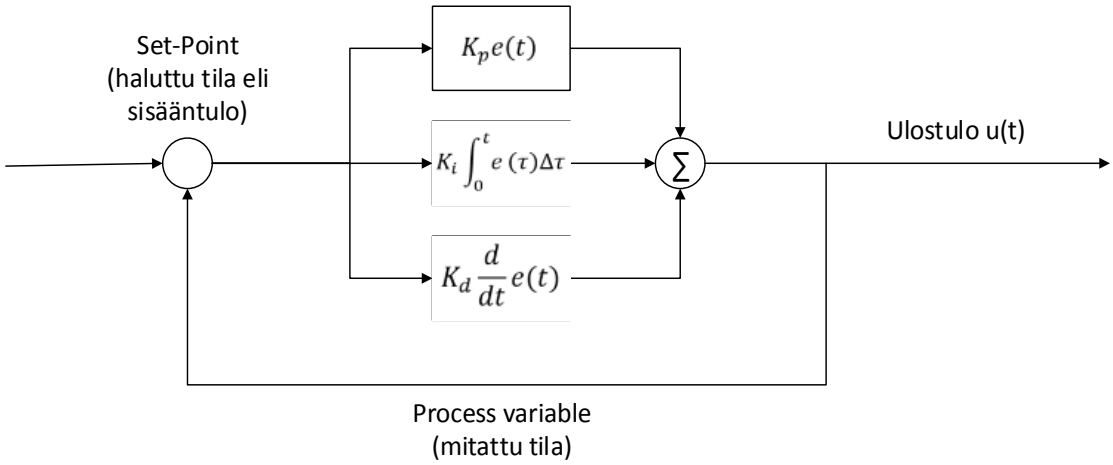
PID-säädin koostuu sisääntulosta sekä em. kolmesta termistä, jotka muodostavat säätimen ulostulon. Nämä kolme termiä kuvaavat muodostavat PID-säätimen nimen sekä kuvaavat sen rakennetta. Termeistä käytetään nimityksiä *Proportional* (summa), *Integral* (integraali) sekä *Derivative* (derivaatta) [16]. Tämän lisäksi säädin vaatii ulostulon takaisinohjaamista säätimen sisääntuloon, muodostaen ns. ”*loopback*” -rakenteen. Yleisesti PID-säätimen ajatellaan noudattavan seuraavaa rakennetta

$$u_t = K_p e(t) + K_i \int_0^t e(\tau)\Delta\tau + K_d \frac{d}{dt} e(t)$$

Missä $K_p$, $K_i$ ja $K_d$ ovat termejään vastaavia säätötermejä, $t$ on aika mittaushetkellä ja $\tau$ on ns. derivaatta-aika, eli mittaukseen kulunut aika joka voidaan laskea kaavalla

$$\tau = t - t_{t-1}$$

Muuttuja $e$ on halutun tilan sekä sen hetkisen tilan välinen erotus. Sitä voidaan myös kutsua virheeksi. Tätä PID-säätimen rakennetta on myös havainnollistavaa kuvata lohkokaaviolla, joka on kuvattuna kuviossa 7.




Kuvio 7. PID-säätimen rakenne.

PID -säätimen sisääntuloa kutsutaan yleisesti englanninkielisellä nimityksellä ”*set-point.*” Voimme käyttää termistä myös lyhennettä *SP*. Sisääntulo määritellään kullekin sovellukselle erikseen. Mitatusta tilasta käytetään yleisesti nimitystä ”*Process variable.*” Voimme käyttää mitatusta tilasta myös lyhennettä *PV*. Havainnollistamisen helpottamiseksi voimme ottaa esimerkkitapaukseksi auton automaattisen nopeudensäätimen.

Tässä tapauksessa sisääntulona PID-säätimelle annettaisiin nopeudensäätimeen haluttu nopeus. Automaattisen nopeudensäätimen tulee pitää nopeus mahdollisimman tarkasti halutussa nopeudessa, mutta kuitenkin pitää nopeus ja kiihdytykset tasaisina. Mikäli PID-säädin olisi huonosti säädetty tai sellaista ei olisi implementoitu, heilahtelisi auton nopeus halutun nopeuden molemmin puolin ajaessa, aiheuttaen nykivää liikettä. PID-säätimen tehtävä on arvioida tarvittavat kiihdytyksen muutokset, jotta nopeus pysyy koko ajan tasaisena [14]. Termi $e(t)$ olisi tässä tapauksessa erotus halutun nopeuden sekä nykyisen nopeuden välillä. Ulostulo *u(t)* olisi tässä tapauksessa tarvittava kiihdytys tai nopeuden hiljennys.

Termi $e$ voidaan ilmaista yleisesti lausekkeella

$$e = SP - PV$$

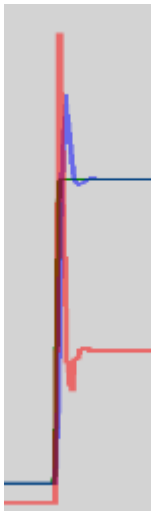
Edellä olevasta lausekkeesta voimme havainnoida PID -säätimen luonteen ns. ”*Loopback*” ohjaimena, sillä edellisen iteraation tulos vaikuttaa seuraavan iteraation tulokseen.

### 3.4.2 PID-säätimen ohjelmointi

PID-säätimen ohjelmoinnissa on tärkeintä ottaa huomioon säätötermien $K_p$, $K_i$ ja $K_d$ vaikutus säätimen toimintaan. Mikäli säätötermit on alustettu huonosti, säädin saattaa esittää epätoivottua toimintaa. Yleensä säätötermien arvot voidaan arvioida kokeellisesti ja havainnoimalla eri säätötermien arvojen muutoksen vaikutus systeemin toimintaan [14 & 16]. Tällä lähestymistavalla päästään suurimmassa tapauksessa applikaatioita tarpeellisen lähelle haluttua tulosta. Termien säätämiseksi optimaaliseen tarkkuuteen on kehitetty myös algoritmeja, joilla säätötermit voidaan optimoida mahdollisimman tarkoiksi halutun toiminnallisuuden saavuttamiseksi [16]. Nämä

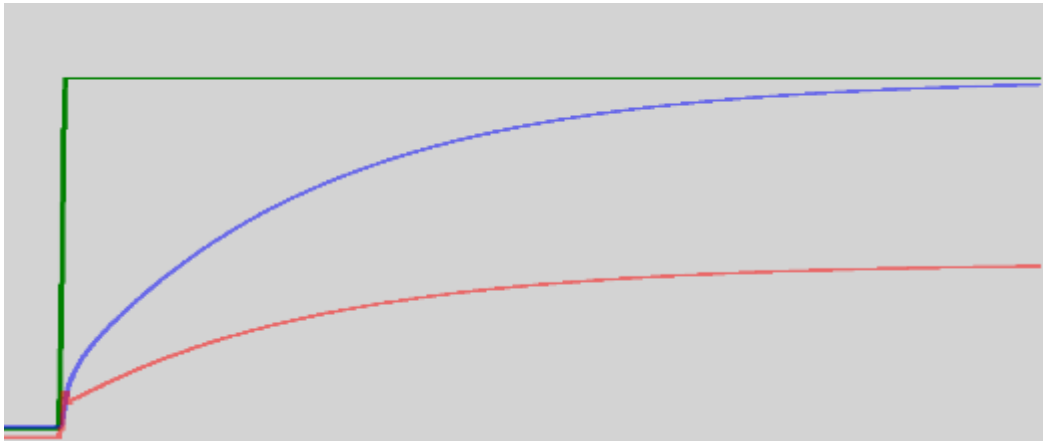
algoritmit vaativat kuitenkin syvällistä tietämystä käytettyjen systeemien toiminnasta sekä ajoituksesta.

Voimme havainnoida eri säätötermien vaikutuksen säätimen ulostuloon kuvioista 8, 9 sekä 10. Kuvaajien vihreä linja on haluttu tila Set-Point, sininen linja on systeemin sisääntulo ja punainen linja on systeemin ulostulo.



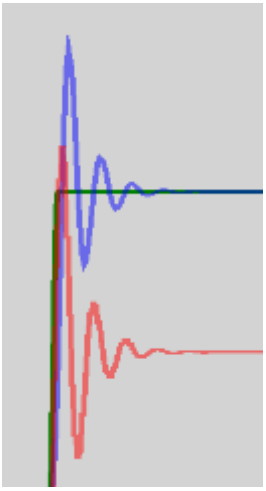
Kuvio 8. Integraalitermin säätötermi liian korkea. Signaali menee yli halutusta arvosta. [14]

Kuviossa 8 on havainnollistettu klassinen tapaus integraalitermin vaikutuksesta säätimen toimintaan. Koska integraalitermi vaikuttaa säätimen äkilliseen toimintaan, se on säädetty liian korkeaksi, jolloin signaali saavuttaa liian korkean arvon tavoitellessaan haluttua ulostuloa, jonka jälkeen se palaa alas. Mikäli integraalitermi on liian alhainen, systeemi reagoi liian hitaasti, joka on havainnollistettu kuviossa 9.



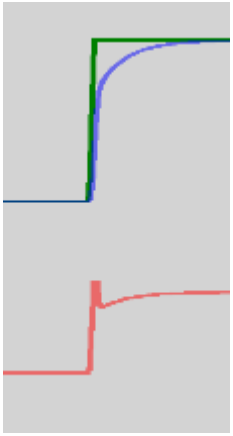
Kuvio 9. Integraalitermi on liian alhainen. Systeemi reagoi liian hitaasti. [14]

Mikäli summatermi tai derivaattatermi säädetään liian korkeaksi, aiheuttaa se signaaliin kaikua, kun ohjain yrittää säätää ulostuloa liian rajusti tavoitellessaan haluttua arvoa. Tätä käyttäytymistä on kuvattu kuviossa 10.



Kuvio 10. Signaali kaikuu ennen kuin se saavuttaa halutun arvon. Derivaatta- ja/tai summatermi liian korkea. [14]

Kuviossa 11 on havainnollistettu PID-säädetyn signaalin käyttäytyminen, kun säätötermit ovat optimoituja.



Kuvio 11. Optimoitu signaali. kaikki säätötermit on asetettu oikeihin arvoihin. [14]

Kuvaajasta näemme, kuinka signaali asettuu pehmeästi haluttuun arvoon, eikä signaali ylitä halutun signaalin arvoa, kuitenkin saavuttaen tilan optimaalisessa ajassa.

Säätimen ohjelmoinnissa tulee myös ottaa huomioon, että mikäli derivaatta- sekä integraalitermi halutaan optimoida, tulee tietää ohjelman iteraatioaika. Aika voidaan yleisesti myös korvata lähellä olevalla vakiolla, mutta täydelliseen optimointiin vaaditaan, että aikaa eri iteraatioiden välillä mitataan.

### 3.4.3 PID-säädin käytännössä

Tässä tutkielmassa PID-säädintä käytettiin yleiskäyttöisen stabilointijärjestelmän moottorinohjaukseen. Moottoreita ohjattiin oikeaan asentoon PID-säätimen kautta, antamalla sille haluttu asentovektori jossa systeemin tulisi pysyä. Tämän jälkeen annettiin sisääntulona mitattu paikkavektori. Moottoreiden tilaa ohjattiin näin haluttuun suuntaan.

Mikäli PID-säädintä ei olisi implementoitu, olisi se aiheuttanut systeemiin paljon häiriöitä moottoreiden nykimisen ja heilahteluiden muodossa, jolloin aktiivinen stabilointi olisi ollut mahdotonta.

PID-säätimen säätötermit mitattiin kokeellisesti tarkkailemalla systeemin toimintaa erilaisilla säätöarvoilla, jonka jälkeen termien arvot uudelleenarvioitiin. Lopputuloksena saavutettiin tarpeellinen tarkkuus yleiskäyttöiseen stabilointiin.

PID-säätimen mahdollistamalla signaalin tasauksella pystytäänkin tasaamaan käytännössä minkä tahansa systeemin toimintaa.

### 3.4.4 Käyttökohteet

PID-säädin kehitettiin alun perin isojen risteilyalusten nopeudensäätimeksi. Nykyään PID-säätimiä voidaan käyttää melkein missä tahansa systeemissä, missä vaaditaan jonkin signaalin tasaista ohjaamista. Tällaisiin systeemeihin lukeutuvat mm. edellä mainittu ajoneuvon automaattinen nopeudensäädin, erilaiset lämpötilasäätimet sekä yleinen moottorinohjaus.

PID-säädin onkin yksi yleisin teollisuudessa käytetty säädin sen suhteellisen yksinkertaisuuden vuoksi, kuitenkin tarjoten paljon käytännöllisyyttä monenlaisiin eri systeemeihin.

# 4 Sensorit

Kolmiulotteisessa tilapaikannuksessa tarvitaan yleisesti ottaen vähintään kuusi vapausakselia järjestelmän toiminnan takaamiseksi. Jotkin järjestelmät toimivat niinkin alhaisella, kuin kolmella vapausakselilla, mutta kolmen vapausakselin järjestelmät törmäävät jossain vaiheessa alhaisen vapausakselilukumäärän tuomiin ongelmiin. Nämä ongelmat ovat luonteeltaan yleensä sensorista riippuvia [7].

Mikäli kolmiakselinen järjestelmä koostuu pelkästään kiihtyvyysanturista, on se altis häiriöpiikeille sekä äkilliset liikkeet tuottavat häiriötä järjestelmään.

Mikäli kolmiakselinen järjestelmä koostuu pelkästään gyroskoopista, on se altis ajelehtimaan ajan myötä.

Joillain kolmiakselisilla järjestelmillä pystytään havainnoimaan kappaleen asentoa yhdellä tai maksimissaan kahdella tasolla, mutta kolmiulotteinen tilapaikannus vaatii onnistuakseen aina vähintään kuusi vapausakselia.

Nämä kolmiakselisen tilanpaikannusjärjestelmän ongelmat pystytään korjaamaan näiden sensoreiden yhdistämisellä, jolloin saamme käyttöömme kuusi vapausakselia. Mikäli paikannuksesta halutaan saada mahdollista jokaisessa olemassa olevassa kappaleen asennossa, on sensoriyhdistelmään lisättävä kolmiakselinen elektroninen kompassi, joka kasvattaa vapausakselien määrän yhdeksään.

## 4.1 Kiihtyvyysanturi tilapaikannuksessa

Kolmiakselinen kiihtyvyysanturi on toinen kolmiulotteisessa tilapaikannuksessa pakollisista sensoreista.

Kiihtyvyysanturi mittaa anturiin kohdistuvia kiihtyvyyden aiheuttamia voimia. Paikallaan oleva kiihtyvyysanturi mittaa painovoimavektorin normaalivoiman ylöspäin. Jos tarkastelemme tätä vektoria pystymme havainnoimaan kappaleen asentoa kolmiulotteisessa avaruudessa. Kiihtyvyysantureita on yksi-, kaksi- sekä kolmiakselisia. Tässä insinöörityössä oletamme anturin olevan kolmiakselinen, kun puhumme kiihtyvyysantureista.

Kiihtyvyysanturi on kuitenkin altis ulkoisten voimien aiheuttamalle värinälle ja tärähtelyille. Tästä syystä sitä ei voida luotettavasti käyttää tilapaikannuksessa ainoana anturina, sillä kappaleen liikkuessa signaaliin muodostuu liikaa häiriöitä luotettavan tuloksen saamiseksi. Tästä syystä kiihtyvyysanturia käytetään lähinnä gyroskoopin arvojen korjaukseen. Gyroskooppi on anturina luotettavampi, mutta ei reagoi nopeisiin liikkeisiin tarpeellisella nopeudella, mutta kiihtyvyysanturi korjaa tämän ongelman. Antureilta saadut arvot yhdistetään erilaisilla fuusioalgoritmeilla.

Vaikka kiihtyvyysanturin mittauksesta otetaan yleensä huomioon eri fuusioalgoritmien luotettavuuskertoimessa vain noin 2-10 %, lisää kiihtyvyysanturi tilapaikannuksen mittausten kokonaisluotettavuutta kuitenkin noin 20–50 %, riippuen systeemistä sekä siihen kohdistuvien muutosten nopeudesta [7]. Tämä kokonaisluotettavuuden lisäys on arvoltaan niin mittava, ettei inertiapaikannussysteemeissä tulisi edes harkita tilapaikannuksen tekemistä ilman kiihtyvyysanturia. Kiihtyvyysanturi on olennainen osa nykyaikaista tilapaikannusjärjestelmää.

## 4.2 Gyroskooppi

Gyroskooppi on toinen kolmiulotteisessa tilapaikannuksessa käytettävistä antureista. Vaikka kappaleen asentoa pystytään arvioimaan pelkällä gyroskoopilla myös kolmiulotteisessa avaruudessa, on siihen liitettävä vähintään kiihtyvyysanturi, koska muuten gyroskooppi kärsii ajan myötä vaihtuvasta arvon ajelehtimisesta.

Gyroskoopista puhuttaessa tämän insinöörityön kontekstissa tarkoitetaan sanalla elektronista gyroskooppia, eikä vanhanaikaista pyörivää analogista gyroskooppia. Gyroskooppi on anturi joka mittaa anturiin kohdistuvaa kulmanopeutta.

Gyroskooppeja on olemassa yksi-, kaksi- sekä kolmiakselisia, mutta tässä insinöörityössä oletamme gyroskoopin olevan kolmiakselinen. Tällainen gyroskooppi pystyy mittaamaan siihen kohdistuvaa kulmanopeutta kolmen eri akselin tai tason suhteen, joka mahdollistaa kaikkien eri asentojen havaitsemisen kolmiulotteisessa avaruudessa.

Kolmiulotteisessa tilapaikannuksessa mittaustulokset pohjautuvat pääasiassa gyroskoopin mittauksiin. Gyroskoopin mittauksia lähinnä korjataan kiihtyvyysanturin mittauksilla sen luontaisen ajelehtimisen korjaamiseksi. Stabiloinnissa suurin osa

anturien luottamusarvoista asetetaan gyroskoopille, sillä se ei ole altis tärähtelyjen aiheuttamille häiriöille, niin kuin kiihtyvyysmittari.

### 4.3 Magnetometer

Magnetomer eli magneettianturi on anturi, joka mittaa siihen kohdistuvien magneettikenttien suuntavektoreita, eli toisin sanoen magneettivuon tiheyttä sekä suuntaa. Olennaista magneettianturin käytössä tilapaikannuksessa on maapallon oman magneettikentän vuon havaitseminen.

Mikäli haluamme esittää jokaisen olemassa olevan asennon kolmiulotteisessa avaruudessa, tulee kappaleen paikan asentoa lukevassa sensoriyhdistelmässä olla yhteensä yhdeksän vapausakselia. Kuudesta vapausakselista siirtyminen yhdeksään hoituu lisäämällä kiihtyvyysanturin sekä gyroskoopin tueksi kolmiakselinen magneettianturi. Vaikka kuudella vapausakselilla pystytään kappaleen asentoa havainnoimaan erittäin tarkasti, poistaa näiden viimeisen kolmen vapausakselin lisääminen tietyt erikoispisteet, joita ei kuusiakselisessa ympäristössä pystytä havainnoimaan. Tämän lisäksi kolme lisäakselia lisäävät mittausarvojen luotettavuutta, sillä kappaleen asentoa pystytään tarkastelemaan myös maan magneettikentän suhteen, eikä navigointi rajoitu pelkästään inertian mittaamiseen.

Magneettianturi kärsii kuitenkin siitä, että sen mittaustuloksia tulee suhteuttaa kiihtyvyysanturilta saatuun asentotietoon, sillä se ei pysty yksin havaitsemaan magneettivuon suuntaa, mikäli vuon vektori on alle 90° kulmassa sen havaitsemisvektoriin nähden. Tämän lisäksi mahdolliset suuret kappaleet, jotka on valmistettu ferromagneettisista aineista, kuten raudasta tai muista metalleista, saattavat aiheuttaa häiriöitä anturin mittaustuloksiin.

Tästä johtuen magneettianturia käytetään lähinnä kiihtyvyysanturin sekä gyroskoopin mittaustulosten kompensoimiseen sekä niiden luotettavuuden kasvattamiseen.

## 5 Yleiskäyttöinen stabilointi sekä tilanpaikannusjärjestelmä

Yleiskäyttöisellä stabilointijärjestelmällä tarkoitetaan elektronista järjestelmää, jonka tarkoitus on tasapainottaa jotain tiettyä kappaletta, käyttäen asennonohjaukseen

erilaisia moottoreita tai muita järjestelmiä, joilla kappaleen fyysistä asentoa pystytään ajamaan haluttuun tilaan. Järjestelmä lukee omaa tilaansa kolmiulotteisessa avaruudessa käyttäen tarkoitukseen soveltuvia antureita, useimmiten gyroskooppeja sekä kiihtyvyysantureita. Tämän tilanpaikannustiedon perusteella järjestelmä korjaa omaa asentoaan kohti haluttua tilaa. Järjestelmän tulee toimia mahdollisimman ”pehmeästi,” eikä se saa olla altis ulkoisille häiriöille, kuten tärinälle, vaan sen tulee reagoida tärinään sekä erilaisiin järjestelmään vaikuttaviin fyysisiin voimiin reaaliaikaisesti.

Yleiskäyttöisyydellä tarkoitetaan, että stabilointijärjestelmää voidaan soveltaa universaalisti erilaisiin sovelluksiin. Järjestelmän tulee pystyä laitteistonsa rajoitusten puitteissa tasapainottamaan erilaisia kappaleita erilaisissa järjestelmissä, kunkin järjestelmän vaatimusten mukaisesti. Järjestelmän tulee olla siirrettävissä eri järjestelmien välillä suhteellisen vähillä muokkauksilla. Järjestelmän tulee myös tukea mahdollisimman montaa erilaista laitteistokokoonpanoa.

Tässä projektissa toteutimme kuvatun yleiskäyttöisen stabilointijärjestelmän, jota tukee kehitetty tilanpaikannusjärjestelmä. Tätä kyseenomaista järjestelmää kuvataan luvussa 6.

Teollisuuden tarve sekä tilanpaikannusjärjestelmille, että stabiloinnille on kasvanut eksponentiaalisesti lähivuosina. Järjestelmiä hyödynnetään jo monella teollisuuden alalla, mutta myös kuluttajille suunnatuissa sovelluksissa.

## 5.1 Tilanpaikannus

Tilanpaikannus on olennainen osa stabilointijärjestelmää. Mikäli stabilointijärjestelmä ei tunne omaa tilaansa kolmiulotteisessa avaruudessa, ei ole mahdollisuutta suorittaa stabilointia muuta kuin näennäisesti. Todellinen kappaleen stabilointi vaatii, että järjestelmän asento tunnetaan. Tilatietoja esitetään tietokonejärjestelmille yleensä asentovektorina, josta käytetty järjestelmä pystyy havainnoimaan oman tilansa suhteessa maapalloon. Näitä tilatietoja hyväksikäytetään joko reagoimalla kappaleen asentoon tai pelkästään mittausmielessä. Mikäli järjestelmä tuntee oman fyysisen tilansa avaruudessa, pystyy se reagoimaan mahdollisiin muutoksiin omassa tilassaan sekä optimoimaan järjestelmän toimintaa automaattisesti, ilman ihmisen erinäistä puuttumista järjestelmän toimintaan.

Tilanpaikannusjärjestelmiä hyödynnetään etenkin ilmailu- sekä avaruusteollisuudessa. Ymmärrettävistä syistä, on erittäin tärkeää tietää esim. lentokoneen asento ilmassa, hyödyntäen erilaisia antureita. Myös huikean kasvun lähivuosina kokenut amatöörilennokkien harrastaminen on erittäin suuri sovellutusalue tilanpaikannusjärjestelmille. Tämän lisäksi tilanpaikannusta hyödynnetään autoiluteollisuudessa, missä mm. autotietokoneet säätävät ajoneuvojen jousituksia antureilta saadun tilatiedon mukaan. Kuluttajille lähimpänä tilapaikannusta soveltava esimerkki ovat modernit älypuhelimet. Ne lukevat omaa asentoansa käyttäen sisäänrakennettuja tilanpaikannusantureita, ja hyödyntävät saatuja anturitietoja mm. kääntäen näyttöä suhteessa puhelimen asentoon. Tilanpaikannus onkin suuri osa nykyaikaisia teknologioita.

Kehitetyssä tilanpaikannusjärjestelmässä hyödynnettiin sensorimittauksia kiihtyvyysanturilta, gyroskoopilta sekä magneettianturilta. Suotimista kehitettiin myös versiot, jotka toimivat ilman magneettianturia. Tilanpaikannuksesta sekä sensorien lukemisesta vastasi mikrokontrolleri, johon sensorit liitettiin.

## 5.2 Stabilointi

Stabilointijärjestelmä on luontainen jatke tilanpaikannusjärjestelmälle. Stabilointijärjestelmän tarkoitus on vakauttaa erilaisia kappaleita eli eliminoida ulkoisten fyysisien voimien vaikutus stabiloitavaan kappaleeseen. Yleisesti elektroniset stabilointijärjestelmät toteuttavat kappaleen vakauttamisen ensin lukemalla kappaleen tilaa ym. tilanpaikannusjärjestelmällä, jonka jälkeen ne reaaliaikaisesti reagoivat kappaleen tilan muutoksiin asennonohjausjärjestelmällä, joka ohjaa kappaleen tilaa moottoreilla haluttuun suuntaan. Yleisimmin kappale halutaan pitää paikoillaan, mutta kappaletta voidaan haluta ohjata kontrolloidusti myös johonkin tiettyyn, systeemin kannalta optimaaliseen asentoon. Stabilointia on ennen elektronisia järjestelmiä harrastettu muun muassa käyttämällä hyväkseen vastapainoja sekä erilaisia riipuspainoja kappaleille. [7]

Kehitetyssä stabilointijärjestelmässä tilanpaikannuksesta vastaava mikrokontrolleri on vastuussa myös kappaleen stabiloinnista. Tilanpaikannusjärjestelmän tuottamat mittaukset ohjataan stabilointijärjestelmälle mikrokontrollerin sisällä, jossa stabilointijärjestelmä päättelee tarvittavat asennonohjaukset, jonka jälkeen stabilointijärjestelmä ohjaa tarvittavat toimenpiteet asennonohjausjärjestelmälle.

Yleisiä stabiloinnin käyttökohteita ovat mm. kamerat. Kameroiden kuvaa pystytään vakauttamaan ohjelmallisesti vain tiettyyn pisteeseen asti, jolloin on luontaista, että kameran stabiliteetti taataan parhaan kuvanlaadun takaamiseksi. Muita käyttökohteita löytyy mm. ilmailu sekä veneilyalalta, joissa stabiliteetti on myös suuressa roolissa.

### 5.3 Asennonohjaus

Asennonohjaus stabilointijärjestelmissä suoritetaan yleensä joko tarkoitukseen soveltuvilla servomoottoreilla tai askelmoottoreilla. Tavalliset DC-moottorit eivät sovellu asennonohjaukseen tarkassa stabiloinnissa, sillä ne eivät pysty antamaan tietoa omasta asennostaan ilman erillistä enkooderia, minkä lisäksi niiden ajaminen tarkkoihin asentoihin on vaikeaa. Tässä projektissa käytettiin asennonohjaukseen modifioituja servomoottoreita. Moottoreita ohjattiin antamalla pulssileveysmoduloitua signaalia, joka vastasi arvoltaan suuntaa sekä nopeutta, mihin moottoria haluttiin ajaa.

Asennonohjaus ei ole aina sidottu stabilointiin. Asennonohjausta tarvitaan melkein jokaisessa teollisuuden alassa, jossa käytetään moottoreita tai muita liikkuvia osia. Varteenotettavimmat alat eksaktissa asennonohjauksessa ovat luultavimmin automaatioteollisuus sekä robotiikka.

## 6 'Koura' stabilointijärjestelmän prototyyppi

### 6.1 Johdanto Kouraan

Koura on projektinimi yleiskäyttöisen stabilointijärjestelmän prototyypille. Laitteen ensimmäisen prototyypin versiolla voitettiin Teknologia’13 -messuilla järjestetyn Wärkkäyskilpailun ensimmäinen sija [19]. Kilpailun järjestäjänä toimi Suomen Tietotekniikan ja Elektroniikan Seura.

Kouran tarkoitus on olla mahdollisimman yleiskäyttöinen järjestelmä stabilointiin. Koura koostuu kahdesta pääosasta. Kummatkin osat pystytään erottamaan toisistaan, näin ollen kasvattamaan laitteen mahdollisia käyttökohteita. Toinen osa on projektiin rakennettu AHRS-systeemi. AHRS-systeemi koostuu mikrokontrollerista sekä kolmesta sensorista. Mikrokontrollerina käytettiin Cyppressin valmistamaa PSoC4-alustan [cy8ckit-042] mikrokontrollerialustaa. Sensoreina Kourassa on kolmeakselinen gyroskooppi, kolmeakselinen kiihtyvyysanturi sekä kolmeakselinen magneettianturi.

Magneettianturina toimi Honeywellin HMC5883L-merkkinen anturi. Gyroskooppina sekä kiihtyvyysanturina käytettiin Invensensen valmistamaa MPU6050-alustaa. AHRS-järjestelmän tarkoituksena on tunnistaa kappaleen asentoa kolmiulotteisessa avaruudessa, mahdollisimman hyvällä tarkkuudella sekä pienellä vasteajalla. Tätä tarkoitusta varten luotiin kaksi eri suodinohjelmistoa, joista toinen käyttää hyväkseen Kalman-suodinta ja toinen Madgwickin suodinta. Suodinten eroja kyseisessä järjestelmässä käsitellään kappaleessa 8.

Toinen laitteen pääosio on moottorinohjausjärjestelmä. Tämän moottorinohjausjärjestelmän tarkoituksena on saada tietoa em. AHRS-järjestelmältä, jonka avulla se stabiloi kappaletta ja yrittää kumota kappaleseen vaikuttavia ulkopuolisia voimia, pitäen sen näinollen tasapainossa. Moottorinohjausjärjestelmä toteutettiin kahdelle akselille käyttäen kahta modifioitua servomoottoria. Moottorien ohjauksesta huolehtii mikrokontrollerialustalle luotu PID-säädin. PID-säädin toteutettiin ohjelmallisesti.

Lisäksi projektiin toteutettiin langaton tiedonsiirto käyttäen ZigBee-verkostoa. ZigBee-moduulina käytettiin Digin valmistamaa Xbee-moduulia. Langattomalla tiedonsiirrolla helpotettiin mittaustulosten saamista mikrokontrollerialustalta.

Kouran ensimmäisellä prototyypillä stabiloitiin kahvikuppia. Tarkoituksena oli estää kahvin läikkyminen käden tärinän tai muiden kahvikuppiin vaikuttavien ulkoisten voimien takia. Kahvikuppia pyrittiin siis pitämään pystysuorassa.

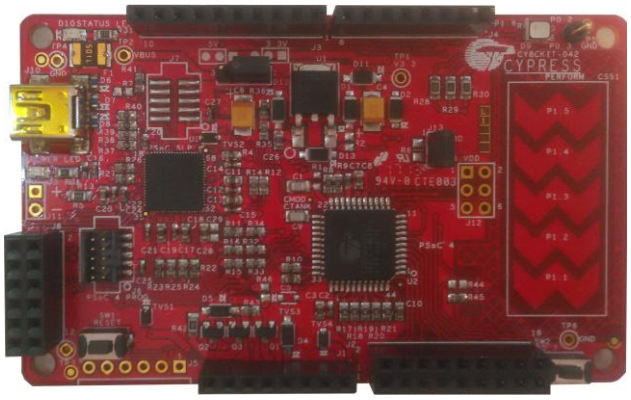
Koura on kuitenkin suunniteltu yleiskäyttöiseksi, ja sillä on moni muita käyttökohteita. Järjestelmää on myös aloitettu sovittamaan quadrokopterilennokin lennonohjausjärjestelmäksi.

## 6.2 Laitteisto

### 6.2.1 PSoC4-kehitysalusta

Kourassa käytetään ”PSoC 4 Pioneer Kit [cy8ckit-042]” -kehitysalustaa. Alustalta löytyy kaksi erillistä mikrokontrolleria. Kuviossa 12 erotellut mikrokontrollerit, joista suurempi

on tyyppiä PSoC 4 ja pienempi PSoC 5 LP. Pääosin alustalla käytetään PSoC 4 mikrokontrolleria, mihin suurin osa tarjolla olevista kytkentäpaikoista on kytketty.



Kuvio 12. [cy8ckit-042] kehitysalusta.

Kuviossa 12 esitetty suurempi mikroprosessori on PSoC 4 ja pienempi on PSoC 4 prosessorin ohjelmoimiseen ja USB-UART-siltaukseen käytetty PSoC 5 LP.

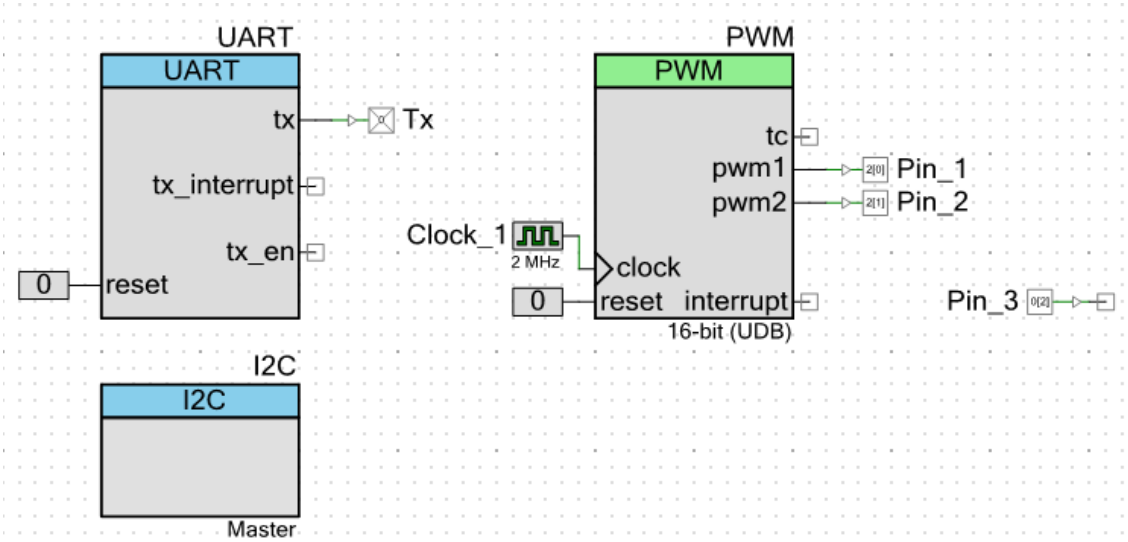
Kourassa käytettään PSoC 4:n ”$I^2C$ master” -moduulia, jota käytetään $I^2C$-väylällä olevien laitteiden lukemiseen. $I^2C$-väylä on asetettu toimimaan MPU6050:n ja HMC5883l maksiminopeudella, joka on valmistajan määrittelemänä 400kHz. $I^2C$-väylän käyttämät pinnit ovat valittavissa vapaista pinneistä.

Servojen ohjaukseen tarvittava signaali muodostetaan PSoC 4 pulssileveysmodulaatio-moduulilla, joka on asetettu toimimaan 50 Hz taajuudella ja pulssileveys on valittavissa 5 % ja 10 % väliltä. Näin luodaan 20 millisekunnin välein toistuva pulssi, jonka leveys vaihtelee 1 millisekunnin ja 2 millisekunnin välillä. PWM-moduulin resoluutio on 16 bittiä, joka takaa riittävän tarkkuuden servomoottorien ohjaukseen.

PSoC 4:n UART-moduulia käytetään debugtulosteiden langattomaan välittämiseen XBee-radion kautta tietokoneelle luettavaksi. UART-moduuli on asetettu toimimaan 115200 baudraten nopeudella.

PSoC 4 prosessorille suunniteltaessa käytetään ”PSoC Creator” -ohjelmaa. Ohjelmalla pystytään määrittämään kaikki mikropiirin käytettävissä olevat ominaisuudet. Kuviossa 13 havainnollistetaan PSoC Creatorin työkalulla haluttujen ominaisuuksien käyttöönottoa.

Kuvio 13. ”PSoC Creator” -ohjelman määritykset prosessorin sisäisille kytkennöille.




Kuviossa 13 pinnit Tx, Pin_1 ja Pin_2 ovat pinnien nimiä, joita käytetään PSoC Creatorin toisessa työkalussa, jossa voidaan valita mihin fyysiseen pinniin signaalit halutaan ohjata.

### 6.2.2 XBee-radiomoduulit

XBee on iDigin valmistama Zigbee-protokollaa käyttävä langattomaan tiedonsiirtoon ja etäohjaukseen suunniteltu laite. Kourassa käytetään kahta XBee moduulia langattomaan UART debugtulosteiden siirtämiseen PC-koneelle. Kuviossa 14 on kuvattuna projektissa käytetty XBEE -moduuli ja sitä varten suunniteltu USB-UART piirilevy.

Kuvio 14. Xbee-radiomoduuli.

XBee-radiomoduulit täytyy asettaa ”point-to-point” -tilaan, jotta debugtulosteet saadaan siirtymään tietokoneelle riittävällä varmuudella. Käytettäessä ”broadcast” -lähetystilaa

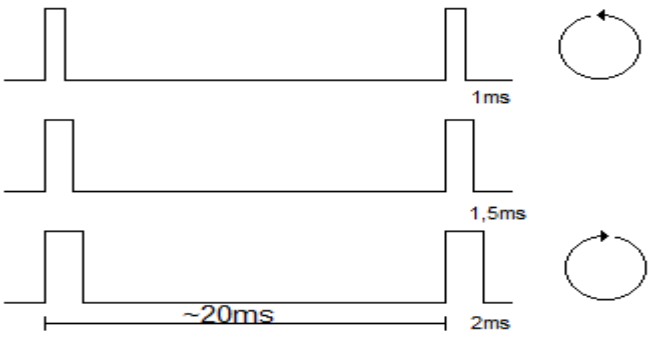
saattaa osa paketeista kadota. ”Point-to-point” -tilassa XBee -moduulit on asetettu lähettämään viestit vain toisilleen, kun taas ”broadcast”-tilassa moduuli lähettää viestipaketit kaikille kantamalla oleville vastaanottimille. Point-to-point saadaan käyttöön asettamalla lähettävän XBee -radiomoduulin ”destination address high” sekä ”destination address low” vastaamaan vastaanottavan XBee-radiomoduulin high ja low -osoitteita.

Xbeen asetuksia voidaan asettaa joko tietokoneella käyttäen mitä tahansa terminaalisovellusta sekä valmistajanmäärittelemiä AT-komentosarjoja tai iDigi:n kyseiseen tarkoitukseen suunniteltua XCTU-ohjelmaa. AT-komentoja käytettäessä asetukset voidaan säätää mikrokontrollerilla.

### 6.2.3 Servomoottorit

Servomoottoria ohjataan pulsseilla vähintään 50 Hz taajuudella. Ohjauspulssin pituus on tavallisesti välillä 1-2ms, mikä vastaa servomoottorin asentoa välillä 0°-180°.

Stabilointijärjestelmää varten servomoottoreita täytyi muokata, jotta saavutettiin haluttu suurempi kuin 180 asteen liikkumavara. Servomoottorin muokkauksessa servomoottorista poistettiin moottorinasennon havaitseva potentiometri. Muokattua servomoottoria ohjataan samalla pulssisuhteella, kuin normaalia servomoottoria, sillä erotuksella, että muokattu servomoottori pysyy liikkumatta pulssin ollessa 1,5 mikrosekunnin pituinen ja pyörii myötäpäivään pulssin ollessa pidempi kuin 1,5 mikrosekuntia sekä vastapäivään pulssin ollessa lyhyempi kuin 1,5 mikrosekuntia. Ohjauspulssin pituudella säädetään servomoottorin suunnan lisäksi moottorin pyörimisnopeutta. Servomoottorin ohjauspulsseja on havainnollistettu kuviossa 15.



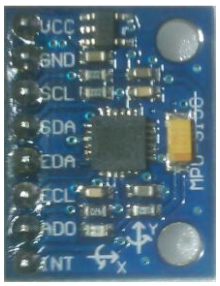

Kuvio 15. Muokatun servomoottorin ohjauspulssit.

Muokkauksen haittavaikutuksena menetetään osa servomoottorin pyörimisnopeudesta, mikä vaikuttaa negatiivisesti tasapainotuksessa asennonkorjauksen nopeuteen. Maksiminopeuden menetyksen haittavaikutus on huomattavissa erityisesti nopeissa ja suurissa liikkeissä, jolloin servomoottorit eivät pyöritä tasapainotettavaa kappaletta riittävän nopeasti ja kappale jää jälkeen.

Servomoottoria tehokkaampi, nopeampi ja tarkempi vaihtoehto on askelmoottori. Askelmoottori vaatii kuitenkin toimiakseen erillisen ohjauspiirin, joka servomoottorin tapauksessa on sisäänrakennettu. Ohjauspiirin aiheuttaman monimutkaisuuden ja servomoottorin ohjauksen yksinkertaisuuden johdosta kouran ensimmäiseen versioon valittiin servomoottorit askelmoottoreiden sijaan.

### 6.2.4 MPU6050

MPU6050 on MEMS-teknologiaan perustuva $I^2C$-väylällä toimiva liikkeentunnistusanturi, joka sisältää kolmiakselisen gyroskoopin, kolmiakselisen kiihtyvyysanturin sekä lämpöanturin. MPU6050 on suunniteltu älypuhelimia, tabletteja ja päälle puettavia antureita varten. Pienen kokonsa ja virrankulutuksensa johdosto sitä käytetään myös radio-ohjattavien kopterien lennonohjauspiireissä. Googlen laseista löytyy MPU9150, mikä eroaa MPU6050:sta siten, että siitä löytyy gyroskoopin, kiihtyvyysanturin ja lämpötila-anturin lisäksi kolmiakselinen magnetometri. Kuviossa 16 on esitetty ”MPU6050 breakout board” -piirilevy.




Kuvio 16. MPU6050 Breakout board.

Kuvan piirilevyltä löytyy MPU6050 ja sen tarvitsemat ulkoiset komponentit, kuten $I^2C$ -väylän ylösvetovastukset ja 3,3 voltin jänniteregulaattori.

MPU6050:ssa on sisäänrakennettu AD-muunnin jokaista luettavaa akselia kohden sekä yksi lämpötilalle. AD-muuntimien tarkkuus on 16 bittiä ja niiden herkkyydet ovat

säädettävissä ohjelmallisesti gyroskoopeille sekä kiihtyvyysantureille erikseen. Kiihtyvyysantureiden AD-muuntimien herkkyydet on valittavissa välillä ±2g ja ±16g, jolloin yhden bitin tarkkuus on väliltä 16384 ja 2048 bittiä/g. Gyroskooppien AD-muuntimien tarkkuus on valittavissa väliltä ±250 – ±2000°/s, jolloin yhden bitin tarkkuus on väliltä 131-164 bittiä /°/s.

MPU6050:n näytteidenotto aloitetaan nollaamalla ”PWR_MGMT_1” -rekisterin SLEEP -bitti, jolloin anturi ottaa näytteitä tasaisella taajuudella. Gyroskooppien näytteidenottotaajuus on 8 kHz, kun anturin sisäänrakennettu alipäästösuodin ei ole käytössä ja 1 kHz kun suodin on käytössä. Kiihtyvyysanturien näytteidenottotaajuus on 1 kHz. MPU6050 voidaan asettaa ilmoittamaan uusista näytteistä, jolloin se asettaa keskeytyspinnin (FSYNC) loogiseen tilaan ”1”. Prototyypissä keskeytyspinni ei ole käytössä, vaan mikro-ohjain on asetettu lukemaan näytteet noin 20 millisekunnin välein.

Kiihtyvyysanturien näytteet on luettavissa rekisteriosoitteista 3Bh–40h. Lämpöanturin lukema on luettavissa rekisteriosoitteista 41h ja 42h. Gyroskooppien näytteet on luettavissa rekisteriosoitteista 43h–48h. Näytteet voidaan lukea MPU6050:n rekistereistä yksi tavu kerrallaan tai ns. ”Burst read” -tavalla, jolloin tavuja voidaan lukea samalla kertaa useampi, noudattaen rekisteriosoitejärjestystä. Kourassa käytetään molempia tapoja. ”Burst read” -tapaa käytetään pääasiassa näytteiden lukemiseen.

MPU6050 on varustettu myös valmistajan omalla Digital Motion Processing -ominaisuudella (DMP), joka voidaan asettaa laskemaan sensorin asentoa, joka puolestaan vähentää mikro-ohjaimen tehovaatimuksia. DMP saadaan käyttöön kopioimalla Invensensen binäärikoodi MPU:n muistiin alustuksen yhteydessä. Kourassa DMP-ominaisuutta ei käytetty, koska Invensensen binäärikoodi itsessään vie suurimman osan PSoC 4 mikro-ohjaimen ohjelmamuistista, jolloin asennonohjausohjelmakoodille ei jää tilaa.

MPU6050:sta löytyvä lämpöanturi on tärkeä, mikäli gyroskoopeilla halutaan päästä valmistajan ilmoittamiin tarkkuuslukemiin, sillä MEMS-gyroskoopit ovat alttiita lämpötilamuutoksille [22]. Kourassa lämpötilakalibrointia ei ole käytetty, sillä testit suoritettiin laboratorio-olosuhteissa.

### 6.2.5 HMC5883l-magneettianturi

HMC5883l on Honeywell-yhtiön valmistama $I^2C$-väylällä toimiva kolmiakselinen MEMS-magneettianturi. Anturi on varustettu kolmella 12 bittisellä AD-muuntimella, jotka mahdollistavat 1–2 asteen kompassisuuntatarkkuuden. Anturin lukemia käytetään korjaamaan gyroskoopin ominaista vaellusta. Kuviossa 17 on kuvattu projektissa käytetty magneettianturi.


Kuvio 17. HMC5883l-magneettianturi.

Kuvion 17 esittämä ”HMC5883L breakout board” -piirilevy on varustettu magneettianturin tarvitsemilla ulkoisilla komponenteilla.

Näytteiden lukua varten magneettianturi tulee anturi asettaa jatkuvan lukemisen tilaan sekä on asetettava anturin käyttämä näytteidenottotaajuus, joka on valittavissa väliltä 0,75 Hz-75 Hz. Lisäksi tulee asettaa anturin vahvistus, joka vaikuttaa näytteiden tarkkuuteen. Asetusten asettamisen jälkeen näytteet ovat luettavissa magnetometrin rekisteriosoitteista 03h–08h. Kouran prototyypissä näytteidenottotaajuus on asetettu noin 20 Hz taajuudelle

## 6.3 Ohjelmisto

Stabilointijärjestelmän ohjaukseen toteutettiin kaksi erillistä ohjelmistoa. Toinen ohjelmisto käyttää asennonpaikantamiseen Eulerpohjaista Kalman-suodinta ja toinen ohjelmisto käyttää kvaterniopohjaista Madgwickin suodinta. Kummatkin ohjelmistot käyttävät asennonkorjaukseen PID -säätimen läpi ajettavaa moottorinohjausta, joka toteutettiin kahdella modifioidulla servomoottorilla. Kalmanpohjaisen ohjelmiston vuokaavio on esitetty kuviossa 18 sekä Madgwickpohjaisen ohjelmiston vuokaavio kuviossa 19.

Ohjelmistot kirjoitettiin C-kiellellä PSoC4-alustalle käyttäen mikropiirivalmistaja Cyppressin PSoC Creator -kehitysympäristöä. Ohjelmistot käyttävät hyväkseen kehitysympäristön valmiita graafisia suunnittelutyökaluja erilaisten tarvittujen sisään- ja ulostulojen ohjaukseen. Sen lisäksi hyväksi käytettiin Cyppressin valmiita kirjastoja

mikrokontrollerialustan ohjaukseen. Lähdekoodin siirtämisen muille laitteille tulisi kuitenkin olla suhteellisen helppoa, sillä ohjelmistoa suunniteltaessa yritettiin koodista kirjoittaa mahdollisimman globaalia, ja Cyppressin omia kirjastoja käytetäänkin lähinnä mikrokontrollerialustan natiivien funktioiden ohjaukseen.

Pääohjelma kutsuu taulukossa 1 lueteltuja tiedostoja sekä niiden sisältämiä funktiota, ja yhdistää eri tiedostojen toiminnallisuuden yhteen ohjelmaan. Ensin pääohjelmassa huolehditaan sekä ohjelmiston, että käytetyn laitteiston alustuksesta. Ohjelmiston alustuksessa kutsutaan nimiketiedostoja (Header), joissa alustetaan ohjelmiston käyttämät tietueet, määritetään funktioiden prototyypit sekä luodaan tarvittavat linkitykset. Yleiskäyttöiset tietueet, funktioiden prototyypit sekä tarvittavat linkitykset määritetään tiedostossa Global.h. Tämä tiedosto sisällytetään jokaiselle ohjelmiston aliohjelmalle. Tämän jälkeen ohjelmistoissa luodaan moottoriohjaukselle haluttu tila (setpoint). Alustuksen seuraavassa vaiheessa kutsutaan `sysInit()` -funktiota, jossa alustetaan tarvittavat laitteistoresurssit.

```
typedef struct sensor
{
            int16 x;
            int16 y;
            int16 z;
} sensor;
```

Esimerkkikoodi 1. Tietue käsittelemättömälle anturitiedolle.

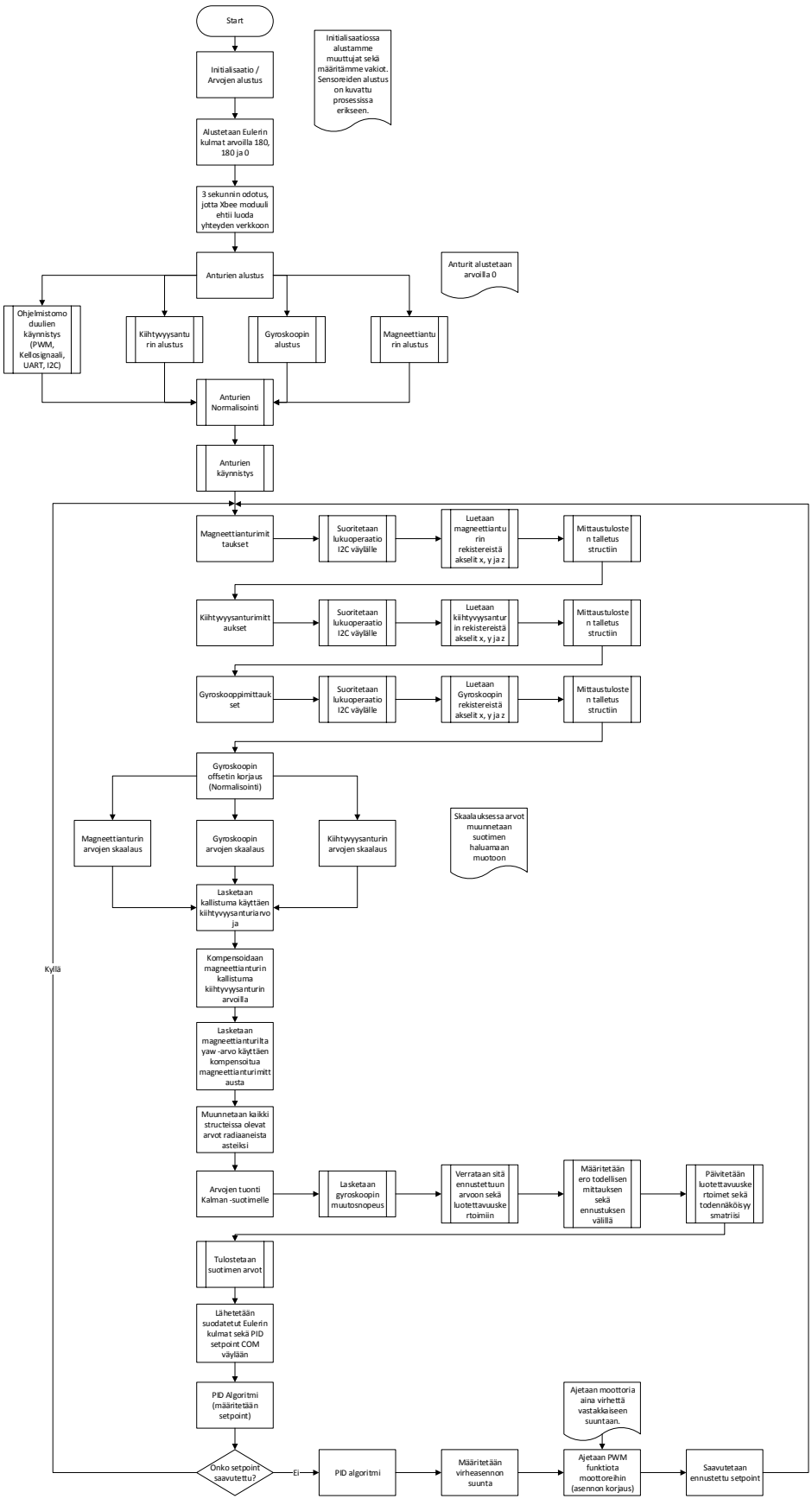
Alustuksen jälkeen pääohjelma alkaa pyörimään kehärakenteessa, jossa käytetään luvussa 6.3.1 käsiteltyjä funktioita, joilla luetaan tietoa antureilta. Antureilta saadut arvot sijoitetaan niille luotuihin tietueisiin, joista tiedot annetaan luvussa 6.3.2 sekä 6.3.3 käsitellyille suodinfunktioille. Näiltä suodinfunktiolta päivitetyt arvot sijoitetaan myös niille määrätyille tietueille. Esimerkki luodusta tietueesta on kuvattu esimerkkikoodeissa 1 ja 2.

```
typedef struct Angles
{
            float yaw;
            float pitch;
            float roll;
} Angles;
```

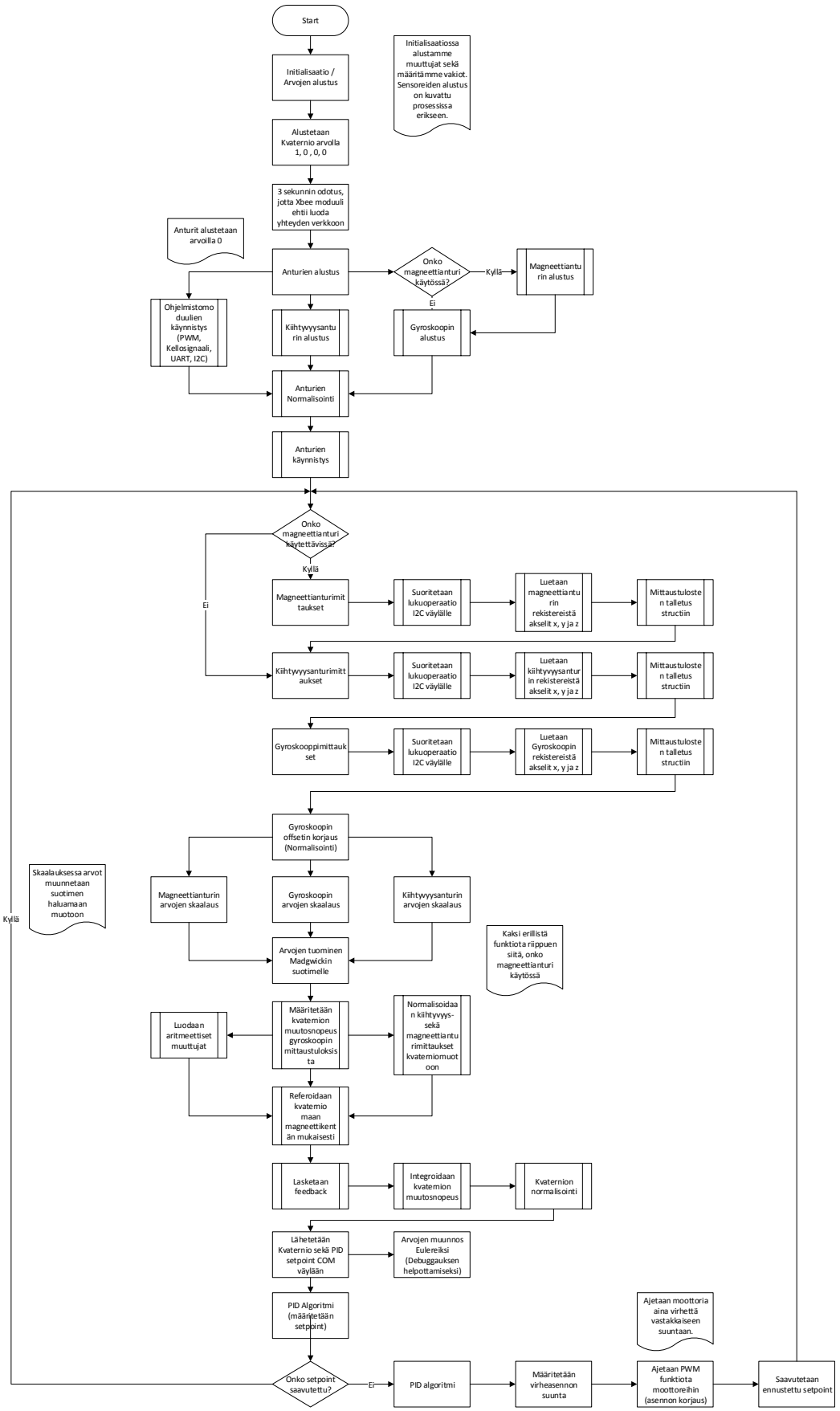
Esimerkkikoodi 2. Tietue suotimella käsitellylle anturitiedolle.

Tietueilta päivitetyt arvot lähetetään sarjaporttiväylälle käyttäen UART -protokollaa. Sarjaporttiväylä on lisätty ohjelmistoon, jotta saatuja arvoja pystytään tarkkailun sekä debuggauksen helpottamiseksi.

Lopuksi arvot annetaan PID-säätimelle, jonka toiminnallisuutta on käsitelty luvussa 6.3.4. PID-säädin huolehtii moottorien ohjaamisesta haluttuun asentoon, sekä moottoreiden käytöksen hallinnasta. Tämän jälkeen pääohjelman kiertorakenne aloitetaan alusta.




Kuvio 18. Kalman-suodinpohjaisen ohjelmiston vuokaavio.




Kuvio 19. Madgwickin suodinpohjaisen ohjelmiston vuokaavio.

Vuokaavioissa on kuvattu ohjelmiston toimintaa kokonaisuutena. Vuokaaviot on helppo rinnastaa ohjelman ns. ”main loopin” toiminnallisuuteen. Main loop on ohjelman päätiedosto, jossa ohjelman suoritusjärjestystä hallinnoidaan. Kyseinen pääohjelma myös huolehtii muiden funktioiden kutsumisesta toisista tiedostoista. Ohjelmisto on jaettu erillisiin lähdekooditiedostoihin, jotta projektia on helpompi hallinnoida. Kaikki projektiin kuuluvat tiedostot sekä selite tiedoston sisällöstä on lueteltu taulukossa 1. Ohjelmiston lähdekoodi on liitteenä.

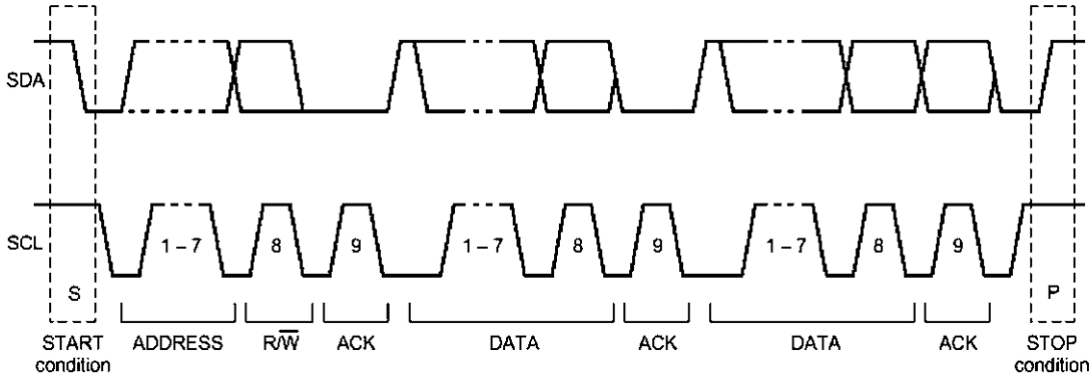
Taulukko 1. Projektitiedostojen selitteet.

| **Lähdekooditiedostot** | **Selite** |
|---|---|
| main.c | Pääohjelma. Huolehtii suoritusjärjestyksestä sekä funktiokutsuista. |
| mahony.c | Sisältää Madgwickin suotimen sekä sen käyttöön tarvittavat funktiot. |
| MPU6050.c | Sisältää MPU6050:n käyttämiseen tarvittavat funktiot |
| itoa.c | Sisältää funktiot joilla voidaan muuntaa integer -tyyppisiä lukuja ASCII tekstiksi. |
| initialization.c | Sisältää ohjelman initialisointifunktiot. |
| HMC5883l.c | Sisältää HMC5883L magneettianturin käyttöön tarvittavat funktiot. |
| kalman.c | Sisältää Kalman-suotimen sekä sen käyttöön tarvittavat funktiot. |
| calculate_angles.c | Sisältää matemaattisia funktioita joita käytetään kulmamuunnoksiin sekä kulmien määrittämiseen |
| **Nimiketiedostot (Header)** | |
| Global.h | Sisältää kaikkien tiedostojen käyttämät vakiot ja yleiskäytössä olevat tietueet (struct) sekä funktioiden prototyyppimääritykset sekä linkitykset. |
| MPU6050.h | Sisältää magneettianturin funktioiden linkityksen |
| mahony.h | Sisältää Madgwickin suotimen käyttämien funktioiden linkityksen sekä suotimen käyttämät vakiot |
| kalman.h | Sisältää Kalman-suotimen käyttämien funktioiden linkityksen sekä suotimen käyttämät vakiot |
| device.h | Sisältää PSoC Creator kehitysympäristön laitekohtaiset määritykset |
| initialization.h | Sisältää alustusfunktioiden linkityksen |
| calculate_angles.h | Sisältää määritykset aritmeettisten kirjastojen käyttöön |

### 6.3.1 MPU6050 IMU:n ohjaus

Lähdekoodi MPU6050 IMU:n ohjaukseen toteutettiin käyttämällä hyväksi Cyppressin natiiveja kirjastoja. Kirjastosta käytettiin Cyppressin natiiveja funktioita kommunikointiin mikrokontollerialustan sekä IMU:n välillä. IMU käyttää kommunikointiin $I^2C$-väylää, joten mikrokontrollerin $I^2C$-ominaisuuksien käyttämiseksi oli tarvittavaa käyttää valmiiksi luotuja funktioita. Muussa tapauksessa olisimme joutuneet kirjoittamaan oman $I^2C$-kirjastomme.

$I^2C$ on Philipsin kehittämä yhden johdon synkroninen sarjaliikenneprotokolla, jossa yksi johtoon kytketty laite toimii ohjaajana ja loput laitteista toteuttavat ohjaajan määräämiä komentoja [18]. Ohjaajasta käytetään nimitystä ”master” ja muita johdolla sijaitsevia laitteita kutsutaan nimellä ”slave.” Master-laite huolehtii kaikesta johdolla käytävästä kommunikaatiosta sekä siitä, etteivät laitteet lähetä tietoja samanaikaisesti. Jokaisella johdolla sijaitsevalla slave-laitteella on oma ID-numeronsa, jonka avulla laitteet tunnistavat niille ohjatut komennot, eivätkä tällöin reagoi muille laitteille tarkoitettuihin komentoihin. Tyypillistä $I^2C$-kommunikointia on havainnollistettu kuviossa 20.




Kuvio 20. Esimerkki $I^2C$ väylän tavanomaisesta kommunikaatiosta. [17]

Rekisterien kirjoittamiseen sekä lukemiseen käytettiin `I2C_I2CMasterWriteBuf()` sekä `I2C_I2CMasterReadBuf()` -funktiota, jossa PSoC4 -alusta toimii masterina. Funktioille annetaan parametreina laitteen $I^2C$ -osoite, luettavat/kirjoitettavat rekisterit sekä rekistereistä luettavien tietojen koko tavuina sekä funktion käyttämä kirjoitustila. Kirjoitustila on Cyppressin natiivi laitekohtainen tila. Funktiot huolehtivat itse tunnistusbittien sekä ACK- sekä NACK-bittien sekä lopetusbittien lähettämisestä.

Tätä funktiota käyttäen loimme omat erilliset funktiomme, jotka ovat tarkoitettu MPU6050 alustan rekisterien lukemiseen, rekistereihin kirjoittamiseen sekä alustan initialisoimiseen. Funktioiden prototyypit ovat

- `mpu6050_Start(void)`
- `mpu6050_read_register(unsigned char *readData, unsigned char reg_address, unsigned char read_count)`
- `mpu6050_write_register(unsigned char reg_address, unsigned char writeData)`

`mpu6050_Start()` -funktio alustaa MPU6050:n käyttöönottoa varten. Alustuksessa laite konfiguroidaan kirjoittamalla halutut arvot erillisiin konfiguraatiorekistereihin. Esimerkkikoodissa 3 on kuvattu esimerkkikonfiguraatio sekä se, miten konfiguraatio kirjoitetaan laitteelle. Funktiolle ei anneta parametreja, ja onnistunut funktion ajo palauttaa arvon nolla.

```
int mpu6050_Start(void) {

            /*Write default config*/
            mpu6050_write_register(mpu6050_reg_pwr_mgmt_1,0x03);  //  POWER  MANAGMENT,
            VALITAAN KELLOKSI GYRON ZAXIS
            mpu6050_write_register(mpu6050_reg_smprt_div,8u);     //    GYRON    OUTPUT
            RATE,  LSKETAAN  KAAVALLA:  kun  dlp_cfg=0,  8kHZ  /  (rekisteriin  kirjoitettu
            arvo (7) +1)
            mpu6050_write_register(mpu6050_reg_config, 6u);
            mpu6050_write_register(mpu6050_reg_gyro_config, 8u);
            mpu6050_write_register(mpu6050_reg_accel_config, 16u);

            UART_PutString("MPU6050 INIT OK...\r\n");
            UART_PutCRLF(0);

            return 0u;
}
```

Esimerkkikoodi 3. MPU6050 konfiguraatiorekisterien kirjoitus.

Koodissa valitaan MPU6050:n käyttämäksi kelloksi laitteen oma sisäinen kide sekä määritetään gyroskoopin laskennallinen nopeus. Muut rekistereihin kirjoitettavat arvot ovat enablointeja sekä niillä määritetään gyroskoopin sekä kiihtyvyysanturin tarkkuusresoluutio.

`mpu6050_read_register` -funktiota käytetään rekisterien lukemiseen. Se käyttää hyväkseen Cyppressin natiiveja funktioita. Funktiolle annetaan parametreina puskuri johon funktio antaa palautusarvot, ensimmäisen luettavan rekisterin osoite sekä luettavien rekisterien lukumäärä. Funktio palauttaa rekistereissä olevat arvot. Funktion sisältö on kuvattu esimerkkikoodissa 4.

```
int mpu6050_read_register(unsigned char *readData, unsigned char reg_address, unsigned
char read_count) {

            I2C_I2CMasterWriteBuf(MPU6050_ADDRESS,&reg_address,1,I2C_I2C_MODE_COMPLETE
            _XFER);

            while(0u == (I2C_I2CMasterStatus() & I2C_I2C_MSTAT_WR_CMPLT));
            I2C_I2CMasterClearStatus();


            I2C_I2CMasterReadBuf(MPU6050_ADDRESS,readData,read_count,I2C_I2C_MODE_COMP
            LETE_XFER);

            while(0u == (I2C_I2CMasterStatus() & I2C_I2C_MSTAT_RD_CMPLT));
            I2C_I2CMasterClearStatus();

            return 0u;
}
```

Esimerkkikoodi 4. MPU6050:n rekistereihin kirjoitukseen käytetty funktio.

`mpu6050_write_register()` -funktiota käytetään rekistereihin kirjoittamiseen. Funktiolle annetaan parametreina halutun rekisterin osoite sekä rekisteriin kirjoitettava arvo. Mikäli kirjoitus on onnistunut, funktio palauttaa arvon nolla. Funktion sisältö on kuvattu esimerkkikoodissa 5.

```
int mpu6050_write_register(unsigned char reg_address, unsigned char writeData) {

    int i, status;
    unsigned char buffer[2];
    buffer[0] = reg_address;
    buffer[1] = writeData;

    status = I2C_I2CMasterSendStart(MPU6050_ADDRESS, I2C_I2C_WRITE_XFER_MODE);

    if(I2C_I2C_MSTR_NO_ERROR == status) /* Check if transfer completed without errors */
    {
        for(i = 0; i < 2; i++)
        {
```

```
            status = I2C_I2CMasterWriteByte(buffer[i]);
            if(I2C_I2C_MSTR_NO_ERROR != status)
            {
              break;
            }
        }
    }
    I2C_I2CMasterSendStop(); /* Send Stop */

    return 0u;
}
```

Esimerkkikoodi 5. MPU6050:n rekisterien kirjoitukseen käytetty funktio.

### 6.3.2 Kalman

Kalman-suotimen ohjelmointi on suunniteltu toimimaan Eulerin kulmia käyttämällä. Ensin tarvittavat raaka-arvot haetaan antureilta käyttäen edellä mainittuja MPU6050 IMU:n ohjaukseen käytettyjä funktioita. Sen jälkeen raaka-arvot muunnetaan käytettävään muotoon joukolla aritmeettisia funktioita. Aritmeettiset toimenpiteet raaka-arvoille suoritetaan `tilt_calc()` -funktiolla. Funktiolle annetaan parametreina raaka-arvot ja funktio palauttaa suodattamattomat kulma-arvot.

Nämä suodattamattomat kulma-arvot ajetaan Kalman-suotimen läpi. Kalman-suodin alustetaan ohjelman alussa omalla funktiollaan. Alustusfunktion prototyyppi on muotoa `void initKalman(kalmanVariables *init_this)`. Funktio ei palauta mitään, mutta parametrina annetaan alustettava Eulerin akseli. Funktio alustaa kaikki Kalman-suotimen käyttämät muuttujat arvolla nolla. Sen lisäksi funktiossa määritellään luotettavuuskertoimien arvot sekä mittausintervalli.

Raaka-arvot annetaan Kalman-suotimelle käyttäen funktiota jonka prototyyppi on muotoa `float kalman_filter(float newAngle, float newRate, kalmanVariables *kVariables)`. Funktio palauttaa sille annetun akselin kulma-arvon suodatettuna. Parametreina funktiolle annetaan kulman raaka-arvo, gyroskoopin raaka-arvo sekä tietue, jossa sijaitsevat päivitetyt luotettavuusarvot halutulle akselille. Esimerkki lähdekoodista on kuvattuna esimerkkikoodissa 6.

```
float kalman_filter(float newAngle, float newRate, kalmanVariables *kVariables)
{
            float S;
            // Luotettavuuskerroin
```

```
            float y;
            // Mittaus miinus arvio mittauksesta
            float rate;

            rate = newRate - kVariables->bias;
            kVariables->angle += kVariables->dt * rate;


            kVariables->P[0][0]   +=   kVariables->dt   *   (kVariables->dt*kVariables-
>P[1][1] - kVariables->P[0][1] - kVariables->P[1][0] + kVariables->Q_angle);
            kVariables->P[0][1] -= kVariables->dt * kVariables->P[1][1];
            kVariables->P[1][0] -= kVariables->dt * kVariables->P[1][1];
            kVariables->P[1][1] += kVariables->Q_gyroBias * kVariables->dt;

            y = newAngle - kVariables->angle;

            S = kVariables->P[0][0] + kVariables->R_angle;

            kVariables->K[0] = kVariables->P[0][0] / S;
            kVariables->K[1] = kVariables->P[1][0] / S;

            kVariables->angle += kVariables->K[0] * y;
            kVariables->bias += kVariables->K[1] * y;

            kVariables->P[0][0] -= kVariables->K[0] * kVariables->P[0][0];
            kVariables->P[0][1] -= kVariables->K[0] * kVariables->P[0][1];
            kVariables->P[1][0] -= kVariables->K[1] * kVariables->P[0][0];
            kVariables->P[1][1] -= kVariables->K[1] * kVariables->P[0][1];

            return (kVariables->angle);
}
```

Esimerkkikoodi 6. Kalman-suotimen lähdekoodi.

Ensimmäisenä funktio päivittää gyroskoopin kulmanopeuden muutoksen vähentämällä siitä lasketun mittausvirheen. Kiihtyvyysanturilta saatuun kulma-arvoon taas lisätään päivitetty kulmanopeus kerrottuna mittausajalla.

Tämän jälkeen päivitetään todennäköisyysmatriisi gyroskoopin arvojen luotettavuuskertoimille. Tämän jälkeen funktio laskee mitatun kulma-arvon ja vertaa sitä ennustettuun arvoon, sekä laskee todellisen arvon näiden parametrien pohjalta. Lopussa funktio päivittää todennäköisyysmatriisin vastaamaan lukuja seuraavalle mittauskierrokselle.

### 6.3.3 Madgwick

Madgwickin suodin on perusluonteeltaan kvaternioihin pohjautuva. Tästä syystä myös ohjelma on kvaterniopohjainen. Madgwickin suotimen ohjelma koostuu pelkästään yhdestä funktiosta, jossa raaka-arvot ajetaan suotimen läpi sekä muutetaan kvaterniomuotoon. Luotua kvaterniota säilytetään sille luodussa tietueessa.

Madgwickin suotimen käyttämä funktio on muotoa `void MadgwickAHRSupdate(float gx, float gy, float gz, float ax, float ay, float az, float mx, float my, float mz)`. Funktio ei anna palautusarvoja, sillä suodin päivittää uuden kvaternioarvon suoraan sille luotuun tietueeseen. Sen lisäksi Kalman-suotimessa olevia aritmeettisia välioperaatioita ei tarvita, sillä suodin toimii käyttäen suoraan antureilta tulevia raaka-arvoja. Funktiolle annetaan parametreina antureiden raaka-arvot jokaiselta akselilta. Mikäli magneettianturia ei ole käytössä, käytetään magneettianturin akselien parametreina arvoja nolla. Esimerkki Madgwickin anturin ohjelmoinnista on kuvattuna tämän dokumentin liitteenä.

Ensin suotimessa lasketaan muutosnopeus käyttäen gyroskoopilta saatuja arvoja. Tämä suoritetaan jokaiselle kvaternion arvolle erikseen oheisen esimerkkikoodin 7 mukaisella tavalla.

```
qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);
qDot2 = 0.5f * (q0 * gx + q2 * gz - q3 * gy);
qDot3 = 0.5f * (q0 * gy - q1 * gz + q3 * gx);
qDot4 = 0.5f * (q0 * gz + q1 * gy - q2 * gx);
```

Esimerkkikoodi 7. Muutosnopeuskvaternion määrittäminen.

Sen jälkeen suodin normalisoi kiihtyvyysanturilta saadut raaka-arvot. Normalisointia on kuvattu oheisessa esimerkkikoodissa 8.

```
recipNorm = invSqrt(ax * ax + ay * ay + az * az);
ax *= recipNorm;
ay *= recipNorm;
az *= recipNorm;
```

Esimerkkikoodi 8. Kulmakvaternion normalisointi.

Tämän jälkeen suoritetaan joukko aritmeettisia operaatioita, joilla luodaan kvaternioarvot kiihtyvyysanturilta saaduilta raaka-arvoilta, sekä referoidaan ne maan magneettikentän mukaisesti. Maan magneettikentän referoimiseksi tarvitaan raaka-arvot magneettianturilta. Tämän jälkeen mittaustuloksia korjataan referoiden ne luotettavuusarvojensa mukaisiin referenssikvaternioihin.

Suodin luo yhden kvaternion kiihtyvyysanturilta saatuihin kulma-arvoihin sekä toisen kvaternion gyroskoopilta saatujen arvojen perusteella, jolla kuvataan kvaternion muutosnopeutta. Tämän jälkeen kvaterniot yhdistetään, jotta saadaan lopullinen tilakvaternio. Tämä suoritetaan esimerkkikoodin 9 mukaisesti.

```
q0 += qDot1 * (1.0f / sampleFreq);
q1 += qDot2 * (1.0f / sampleFreq);
q2 += qDot3 * (1.0f / sampleFreq);
q3 += qDot4 * (1.0f / sampleFreq);
```

Esimerkkikoodi 9. Muutosnopeuskvaternion sekä kulmakvaternion yhdistäminen.

Lopuksi saatu kvaternio päivitetään sille luotuun tietueeseen. Kvaterniolle määritetty tietue on kuvattu esimerkkikoodissa 10.

```
typedef struct Quaternions
{
            float w;
            float x;
            float y;
            float z;

} Quaternions;
```

Esimerkkikoodi 10. Kvaterniotietue.

### 6.3.4 PID-säädin

PID-säädin on sisällytetty pääohjelmatiedostoon. Säädin toimii yhden funktion kautta, jonka prototyppi on muotoa `float PID (int setpoint, float angle, float error, float prevError, float integ)`. Funktio palauttaa arvona moottorinohjaukseen käytettävän PWM-arvon. Tätä PWM-arvoa käytetään suoraan

moottoreiden ohjaamiseen. Funktion parametrit ovat haluttu arvo (setpoint), nykyinen kulma, edellisellä kierroksella havaittu virhe nykyisen tilan sekä halutun tilan välillä, nykyinen virhe, sekä integraatiotermi. PID-säätimen funktio on kuvattu esimerkkikoodissa 11.

```
float PID (int setpoint, float angle, float error, float prevError, float integ)
{
    float output;
    float deri;

    error = setpoint - angle;
    integ += error;
    deri = (error - prevError);

    output = (KP*error) + (KI * integ) + (KD * deri);

    prevError = error;

    return (output);
}
```

Esimerkkikoodi 11. PID -säädinfunktio.

Funktio määrittää ensin eron halutun tilan sekä nykyisen tilan välillä, jonka jälkeen lasketaan summa-, derivaatta- sekä integraatiotermien arvo käyttäen ennalta määrättyjä vakioita. Tämän jälkeen lasketaan funktion palauttama moottorinohjausarvo.

Lopuksi funktio päivittää nykyisen virhearvion, jotta se on käytettävissä, kun funktiota kutsutaan seuraavan kerran.

# 7 LabVIEW testausohjelmisto

Projektia varten toteutettiin kaksi erillistä testausohjelmistoa National Instrumentsin LabVIEWllä. LabVIEW on graafinen ohjelmointikieli, joka on suunniteltu eritoten mittauksia sekä automaatiota varten. LabVIEWllä toteutettiin erilliset ohjelmistot yleiskäyttöisen stabilointijärjestelmän testaukseen sekä suodinvertailun testaukseen. Ohjelmistot suunniteltiin kummankin skenaarion tarpeiden mukaisesti. Ohjelmistojen lähdekoodit eli ns. ”Block diagrammit” ovat liitteenä.

## 7.1 LabVIEW -ohjelmisto stabilointijärjestelmälle

Ohjelmisto toteutettiin yleiskäyttöisen stabilointijärjestelmän kehitysvaiheessa. Ohjelmiston tarkoituksena oli auttaa stabilointijärjestelmän kehityksessä, tarjoten analysointivaihtoehtoja eri antureilta saaduille mittauksille, sekä auttaa niiden hahmottamisessa tosielämän tilanteessa. Kehitysympäristönä käytettiin LabVIEWn 2012 SP1 versiota. Ohjelmiston ajamiseen ei tarvita muita lisämoduuleita. Ohjelmistoon käytettiin perusversioon kuuluvia 3D Graphics- sekä 3D Picture Control -funktiokirjastoja.

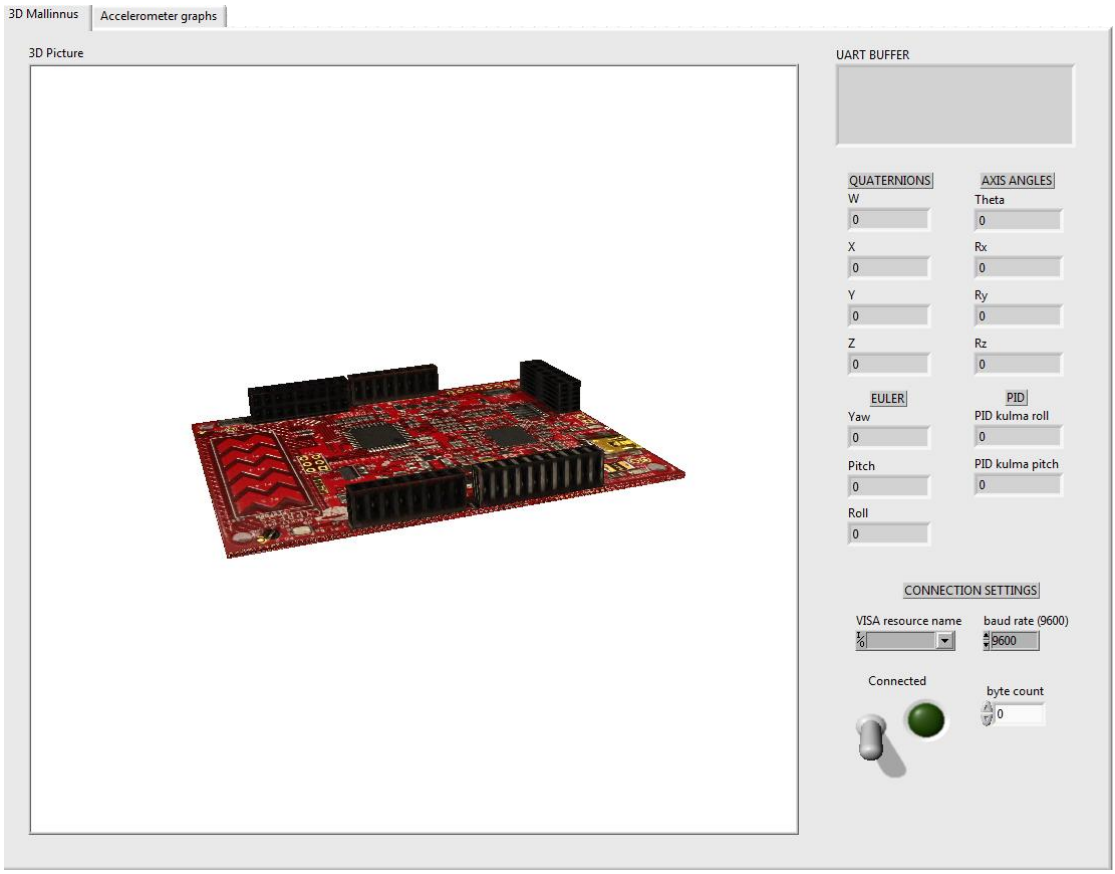
Ohjelmisto lukee erilaisia muuttujia käyttäen RS-232-sarjaporttiprotokollaa. Muuttujat lähetetään ohjelmalle sarjaportin kautta suoraan stabilointijärjestelmää ohjaavalta mikrokontrollerilta. Mikrokontrolleri tulostaa sensorilta saadut arvot raakadatana, sekä toimittaa niistä lasketut kvaterniot LabVIEW ohjelmistolle. Mikäli mikrokontrolleri käyttää toimintaansa Kalman-suodinta, tulostaa se sarjaväylälle kvaternioiden sijaan Eulerin kulmia. LabVIEW-ohjelmisto lukee nämä tiedot sarjaväylältä, ja pystyy datan esitysmuodosta päättelemään käyttääkö mikrokontrolleri esitysmuotonaan kvaternioita vai Eulerin kulmia. Mikrokontrolleri lähettää anturitiedot sekä käsitellyt muuttujat taulukossa 2 määriteltynä viestipakettina, riippuen käytössä olevasta suotimesta.

Taulukko 2. Mikrokontrollerin lähettämä viestipaketti stabilointijärjestelmältä.

| **BYTE** | **Kalman** | **Madgwick** |
|---|---|---|
| **1** | Tunniste | Tunniste |
| **2** | Pitch | q.w |
| **3** | Roll | q.x |
| **4** | Yaw | q.y |
| **5** | PID Pitch | q.z |
| **6** | PID Roll | PID Pitch |
| **7** | N/A | PID Roll |

Viestipaketti on käytössä olevan suotimen valinnasta riippuen joko kuusi- tai seitsemäntavuinen. Ensimmäinen tavu on varattu sisäiselle tunnistustavulle kummassakin vaihtoehdossa. Tunnistustavulla määritellään käytössä oleva tilavektorin esitysmuoto. Mikäli käytössä on Kalman-suodin, tavut 2-4 on varattu Kalman-suotimen tuottamille Eulerin kulmille, kuten on kuvattu taulukossa 2. Kalman-suotimen ollessa käytössä, seuraavat kaksi viimeistä tavua on varattu PID -säätimen mittaamille pitch- ja

roll arvoille. Mikäli taas käytössä on Madgwickin suodin, on käytössä seitsemän tavua. Tavut 2-5 on varattu Madgwickin suotimen tuottamille kvaternion arvoille, kuten on kuvattu taulukossa 2. Viimeiset kaksi tavua on taas varattu PID-säätimen mittaamille kulma-arvoille.




Kuvio 21. LabVIEW-testausohjelmiston Front Panel.

LabVIEW-ohjelmistolla pystytään seuraamaan sensorimittausten vaikutusta jo käsiteltyihin arvoihin ihmiselle luettavassa muodossa. Ohjelmisto myös mallintaa kolmiulotteisen kuvan anturin asennosta, käyttäen LabVIEWn valmiita funktioita 3D-mallinnukseen. Näin pystyttiin reaaliaikaisesti vertailemaan anturin todellista asentoa asentoon, jossa ohjelmisto tulkitsee sen olevan. Tämän lisäksi pystyttiin vertailemaan suodinten käyttäytymistä erilaisilla säätöarvoilla verraten mallinnetun 3D-kuvan käyttäytymistä sekä mikrokontrollerilta saatuja numeerisia arvoja anturin todelliseen tilaan. Kehitetyn LabVIEW-ohjelmiston front panel on kuvattu kuviossa 21.

Ohjelma luo ensin kolmiulotteisen tilanteen (ks. englannin ”scene”), jonka jälkeen se rakentaa 3D-kuvan mikrokontrollerilaudasta 3D-objektina. Tämän jälkeen ohjelma

alustaa UART-sarjaliikenneväylän toimintaa, käyttäen LabVIEW VISA -funktiota. UART-liikenteen määritykset on kuvattu taulukossa 3.

Taulukko 3. Sarjaliikennemääritykset.

| **SUURE** | **ARVO** |
|---|---|
| **Resource name** | Tulee olla käytetty COM -portti. Säädettävissä Front Panelista. |
| **Baudrate** | Säädettävissä Front Panelissa. Tulee vastata mikrokontrollerin baudratea. Käytetty baudrate oli 115200 |
| **Parity Bit** | None |
| **Stop bits** | 1.0 |
| **Flow Control** | None |
| **Termination Char** | 0xA / 10 |

Tämän jälkeen ohjelmisto aloittaa lukemaan liikennettä sarjaväylältä. Tietoa luetaan taulukon 2 mukaisina pakettiviesteinä. Mikrokontrollerin tulostusfunktioiden johdosta (mikrokontrolleri ei pysty tulostamaan floating point -lukuja sarjaliikenneväylään) kvaternioiden arvot on kerrottu tuhannella, joten ohjelma jakaa saadut kvaterniot taas tuhannella. Tämän jälkeen mikrokontrollerilta saaduista arvoista muodostetaan klusteri.

Klusterilla ohjataan alustusvaiheessa muodostettua 3D-objektia käyttäen Set Rotation VI -funktiota. Tämän jälkeen 3D-objektin asento piirretään näyttöruudulle. Klusterissa sijaitsevan kvaternion arvoista lasketaan myös vastaavat arvot Eulerin kulmille sekä vektorille ja kulmalle käyttäen vastaavia muunnosalgoritmeja. Nämä arvot tulostetaan Front Paneliin käyttäjälle, jotta arvoja pystytään vertailemaan.

Mikäli mikrokontrollerin käytössä on Kalman-suodin, tehdään em. toimet mukautettuna niin, että lähtöarvot ovat Eulerin kulmia, joista lasketaan muunnosarvot vektorille ja kulmalle sekä kvaterniolle.

## 7.2 LabVIEW-ohjelmisto suodinvertailussa

Suodinvertailuun käytetty LabVIEW-ohjelmisto rakennettiin samalle ohjelmapohjalle kuin stabilointijärjestelmän tueksi rakennettu ohjelmisto. Sen toiminnallisuutta kuitenkin muokattiin niin, että sarjaportilta saatuja arvoja pystyttiin kirjoittamaan lokiin, sekä sillä pystyttiin analysoimaan arvoja lokitiedostoista. Mikrokontrollerin lähettämä viestipaketti suodinvertailun ohjelmistolle on kuvattuna taulukossa 4. Suodinvertailua suorittavalle

ohjelmistolle rakennettiin myös ylimääräisiä analysointityökaluja suotimilta saatavien signaalien analysointiin. Signaalit havainnollistettiin perinteisillä XY -taulukoilla.

Taulukko 4. Mikrokontrollerin lähettämä viestipaketti suodinvertailun testiohjelmistolta.

| **BYTE** | **Kalman** | **Madgwick** |
|---|---|---|
| **1** | Tunniste | Tunniste |
| **2** | Pitch | q.w |
| **3** | Roll | q.x |
| **4** | Yaw | q.y |
| **5** | PID Pitch | q.z |
| **6** | PID Roll | PID pitch |
| **7** | Gyro x | PID roll |
| **8** | Gyro y | Gyro x |
| **9** | Gyro z | Gyro y |
| **10** | Acc x | Gyro z |
| **11** | Acc y | Acc x |
| **12** | Acc z | Acc y |
| **13** | Mag x | Acc z |
| **14** | Mag y | Mag x |
| **15** | Mag z | Mag y |
| **16** | N/A | Mag z |

Kuten taulukosta 4 saatetaan havaita, noudattaa mikrokontrollerin lähettämä viestipaketti Kalman-suotimen ollessa käytössä tavuilta 1-6 sekä Madgwickin suotimen ollessa käytössä tavuilta 1-7 samaa rakennetta kuin stabilointijärjestelmän lähettämä viestipaketti. Testausohjelmiston lähettämä viestipaketti on kuitenkin rakenteeltaan joko 15- tai 16-tavuinen. Kalman-suotimen ollessa käytössä, on viestipaketti 15 tavua, ja vastaavasti Madgwickin suotimen ollessa käytössä, on viestipaketin koko 16 tavua. Viestipaketin loppuun on testiohjelmassa lisätty raakadata gyroskoopilta, kiihtyvyysanturilta sekä magneettianturilta.

Mikäli käytössä on Kalman-suodin koostuvat tavut 7-9 gyroskoopin eri akselien raaka-arvoista akseleilta x, y sekä z, taulukon [4] mukaisesti. Vastaavat tavut Madgwickin suotimen ollessa käytössä ovat 8-10. Tavut 10–12 ovat Kalman-suotimen ollessa käytössä vastaavat raaka-arvot kiihtyvyysanturille, sekä tavut 12–15 vastaavien akselien arvot magneettianturilta kuten on kuvattu taulukossa 4. Vastaavat tavut Madgwickin suotimen ollessa käytössä ovat kiihtyvyysanturille 11–13 sekä magneettianturin akseleille 14–16.

# 8 Suodinten vertailu

Suotimia vertailtiin ottamalla näytteitä suotimien antamista arvoista 100 Hz näytteenottotaajuudella. Erilaisia mittauksia suoritettiin noin kymmenen kappaletta. Jokaisessa mittauksessa simuloitiin erilaisia liikkeitä. Liikkeet suoritettiin käyttämällä luvussa 6 esiteltyä mittauskorttia. Mittauskorttia käännettiin käsin erilaisten liikkeiden aikaansaamiseksi, sekä niistä tallennettiin anturien antamat raaka-arvot. Tämän jälkeen raaka-arvot ajettiin suotimien läpi ja tallennettiin tekstitiedostoihin. Näin saatiin aikaiseksi kattava testikokonaisuus erilaisille skenaarioille sekä pystyttiin vertailemaan suotimien suorituskykyä samasta liikkeestä rakentamatta erillistä monimutkaista testijärjestelmää.

## 8.1 Testijärjestelmä

Testijärjestelmä koostui yhdestä Raspberry PI -kehitysalustasta sekä luvussa 6 esitellystä MARG-laudasta. Testeissä analysoitiin pelkästään suodinten suorituskykyä, joten testeissä ei tarvittu ottaa huomioon kappaleen stabilointia, eikä näin ollen PID-säädintä tai sen vaikutusta järjestelmään.

Näytteet otettiin MARG-laudalta, käyttäen luvussa 6 esiteltyä ohjelmistoa muokattuna toimimaan Raspberry PI -alustalle. Näytteistä tallennettiin raaka-arvot, jotka ajettiin kummankin suotimen läpi. Suotimen läpi ajetut arvot, sekä niitä vastaavat raaka-arvot tallennettiin lokitiedostoon. Näytteet analysoitiin lokitiedostosta erillisellä LabVIEW-ohjelmalla.

### 8.1.1 Ohjelmisto

Näytteidenottoa varten ohjelmisto mukautettiin toimimaan Raspberry PI –ohjelmointialustalle. Käyttäen hyväksi ohjelmointialustan Linux-käyttöjärjestelmää saatiin antureilta otetut näytteet tallennettua lokiin käyttöjärjestelmän valmiilla työkaluilla.

Näytteiden analysointia varten rakennettiin analysointiohjelma LabVIEWllä. Analysointiohjelma lukee luotuja lokitiedostoja, erottelee tiedot lokista sekä piirtää saaduista näytteistä kuvaajat. Ohjelma myös erottelee lokitiedoston näytteistä raaka-arvot anturin mukaan sekä piirtää kuvaajat halutuista arvoista. Ohjelma myös piirtää

lokitiedostossa toteutetun liikkeen kolmiulotteisena mallina. Ohjelmassa luodaan mallia varten kaksi erillistä kuvaa. Toinen malli kuvaa Kalman–suotimen läpi annettua arvoa sekä toinen Madgwickin suotimen läpi ajettua arvoa.

### 8.1.2 Näytteenkeruu

Näytteitä kerättiin 100 Hz taajuudella MARG-alustalta. Tämän taajuuden laskettiin vastaavan Nyquistin teoreemaa [20] MARG-alustalta saatavan näytteistysnopeuden mukaisesti.

Näytteitä kerättiin kahdenkymmenen sekunnin ajanjakso ym. näytteenottotaajuudella. Näin saatiin keskenään vertailukelpoisia sekä yhdenmukaisia otteita suodinten toiminnasta. Eri näytekokonaisuuksia otettiin erilaisista käsin suoritettavista koetilanteista. Koetilanteissa MARG-alustaa liikutettiin käsin ennalta suunnitellun mallin mukaistesti. Koetilanteet suunniteltiin etukäteen testaamaan sekä suotimia erilaisissa tilanteissa, mutta myös tuomaan esille suodinten eroja erilaisissa koetilanteissa. Näistä koetilanteista valittiin jatkoanalyysiin neljä erilaista liikettä.

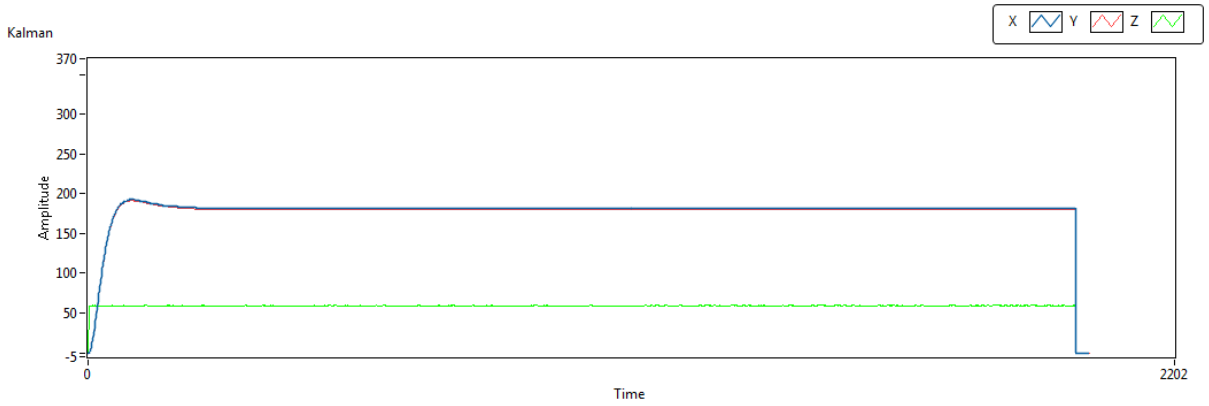
Vaikka näytteitä varten suoritetut liikkeet tehtiin käsin, saatiin tuloksista vertailukelpoisia, sillä vertailuarvot kummallekin suotimelle johdettiin antureilta saaduista raaka-arvoista. Samoista raaka-arvoista johdettiin tulokset kummankin suotimen läpi. Näin eliminoitiin ihmiskäden suorittaman liikkeen epätarkkuus, joka olisi tullut ilmi tuloksissa, mikäli arvot olisi johdettu suoraan suotimien läpi kahdesta erilaisesta liikkeestä. Käsin suoritettujen liikkeiden etuna mainittakoon, että käsin pystyttiin suorittamaan myös monimutkaisia sekä monia erilaisia testitilanteita ilman erillisen testialustan rakentamista sekä testeihin oli helppo luoda luonnollisenoloista häiriötä, kuten tärinää sekä kohinaa.

## 8.2 Tulokset

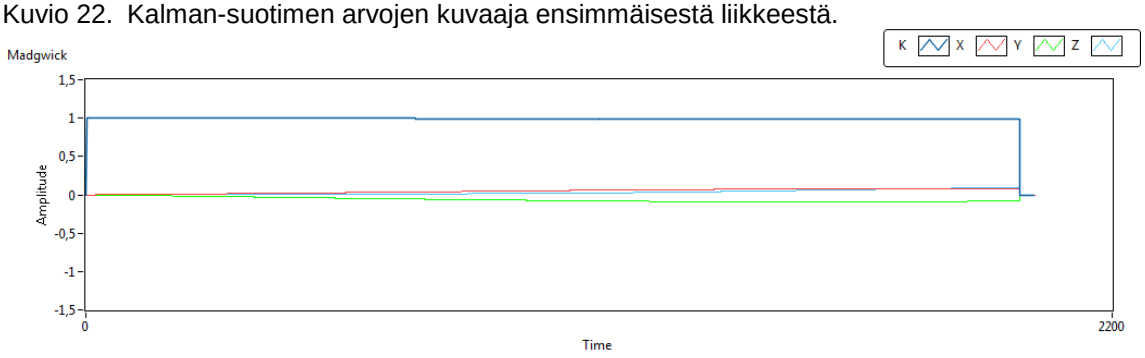
Kerätyistä näytteistä valittiin analysoitavaksi viisi erilaista liikettä. Näytteet valittiin, koska niiden kuvaamilla liikkeillä pystyttiin havainnollistamaan suotimien välisiä eroja eri skenaarioissa.

Ensimmäisessä liikkeessä sensorit pidettiin paikallaan kahdenkymmenen sekunnin ajan. Tällä liikkeellä pyrittiin havainnoimaan anturien antamista näytteistä mahdollinen

ajelehtiminen, sekä tarkastamaan suotimien kykyä suodattaa varsinkin gyroskoopin ominainen ajelehtiminen mittaustuloksista. Ensimmäisen suoritetun liikkeen näytteistä johdetut kuvaajat on esitetty kuvioissa 22 ja 23.




Kuvio 22. Kalman-suotimen arvojen kuvaaja ensimmäisestä liikkeestä.



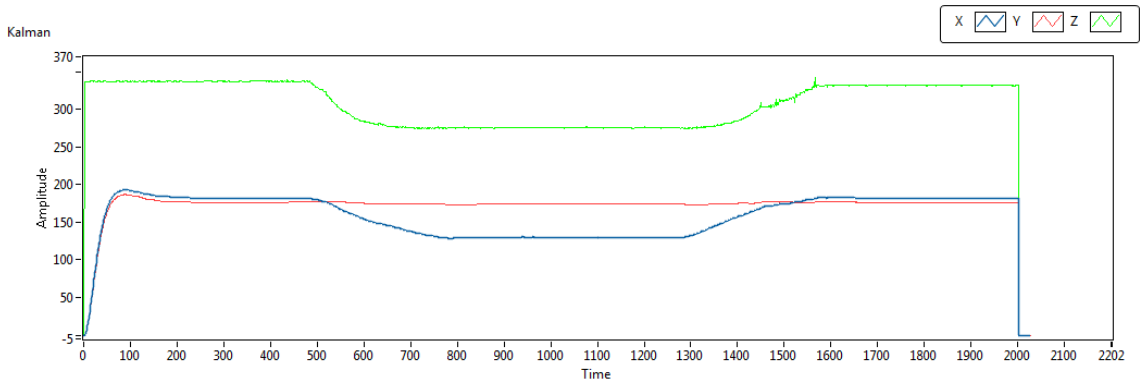

Kuvio 23. Madgwickin suotimen arvojen kuvaaja ensimmäisestä liikkeestä.

Ensimmäisistä näytteistä pystyttiin huomaamaan Magwickin suotimessa pientä ajelehtimista z-akselin ympäri. Ajelehtiminen kesti kuitenkin vain noin ensimmäisen viiden sekunnin ajan, jonka jälkeen arvot normalisoituivat. Alun arvojen ajelehtiminen pystytään selittämään suotimen säätötermien arvoilla, jotka eivät välttämättä olleet kokeen aikana optimoituja.

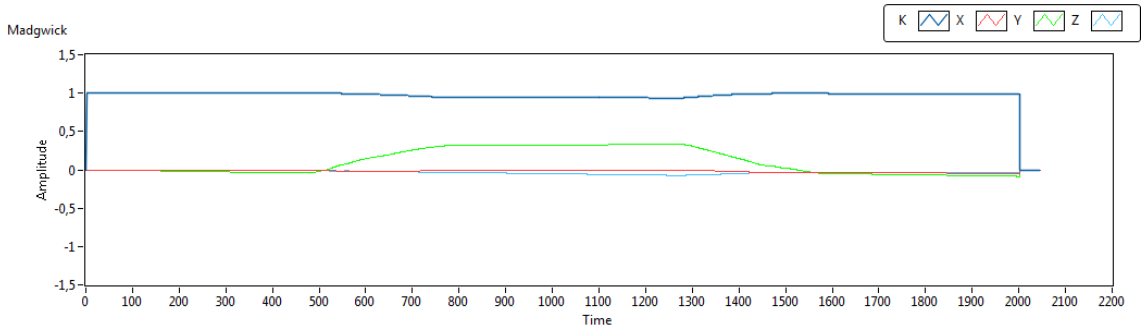
Kalman-suotimessa sen sijaan on havaittavissa jokaisen näytetiedoston alussa noin sekunnin mittainen alustusjakso, jossa arvot hakeutuvat nykyiseen arvoon. Tämä selitetään Kalman-suotimen luomien luotettavuuskertoimien alustuksella. Arvot on alustettu arvoon 0, jolloin luotettavuuskertoimet tulee johtaa, ennen kuin suotimelta saadaan käyttökelpoisia arvoja. Näytteissä tämä havaitaan tällä pienellä alustusajalla.

Kumpikin suodin suoriutui ensimmäisestä koetilanteesta hyvin, eikä huomattavaa häiriökäyttäytymistä ollut havaittavissa.

Toisessa koetilanteessa sensorialustaa liikutettiin noin 45 asteen kulmaan jokaiselle akselille erikseen. Nämä kallistukset lautaan tehtiin vaiheittain, palauttaen sensorialusta jokaisen kallistuksen jälkeen takaisin alkuasentoon ennen seuraavaan kulmaan kallistamista. Toisessa koetilanteessa ei suodinten välillä ollut huomattavia eroja. Kuvaajat toisen koeliikkeen näytteistä on havainnollistettu kuvioissa 24 ja 25.




Kuvio 24. Kalman-suotimen arvojen kuvaaja toisessa liikkeestä.



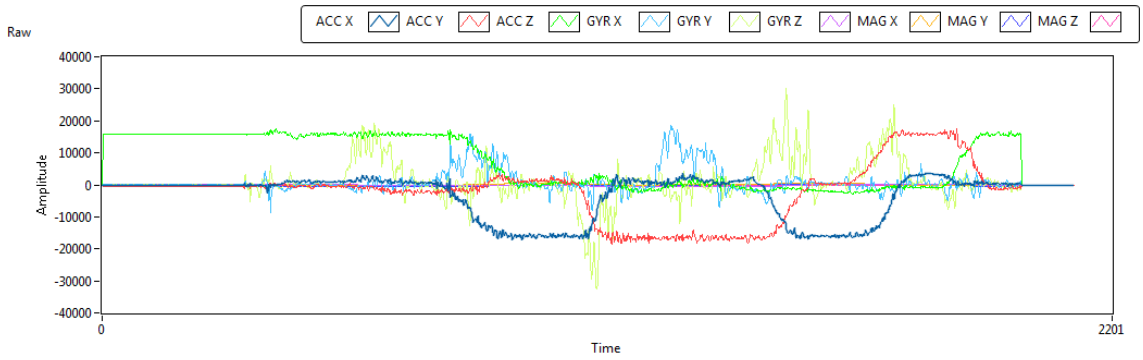

Kuvio 25. Madgwickin suotimen arvojen kuvaaja toisesta liikkeestä.

Kuvaajista ei pystytä havainnoimaan huomattavaa virhekäyttäytymistä ja suotimien antamat arvot ovat stabiileja.

Kolmannessa kokeessa sensorialustaa liikutettiin yli 90 asteen. Ennakkoon odotettavissa oli Kalman-suotimen heikompi suoriutuminen kokeista, joissa sensorialustaa liikutettiin 90 akselin kulmaan yhden akselin mukaisesti, jonka jälkeen

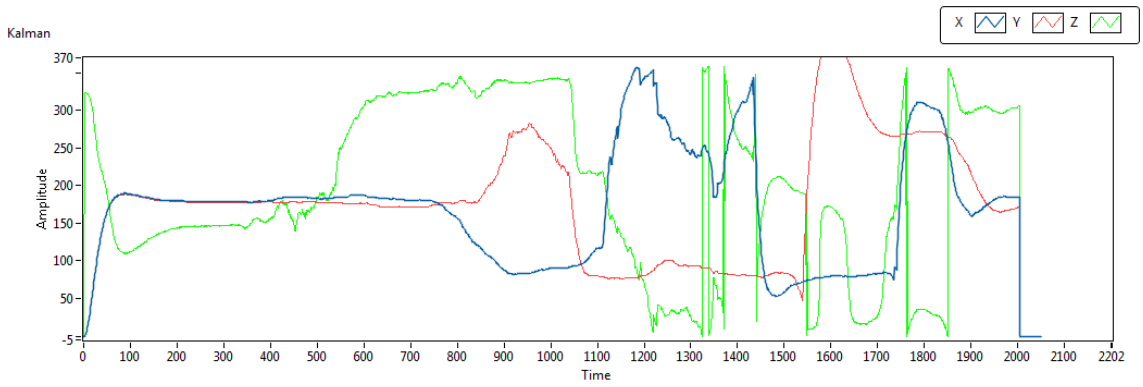
lautaa käännettäisiin toisen akselin mukaisesti. Nämä ennakko-odotukset johtuivat Kalman-suotimen käyttämän Eulerin kulmien ominaisesta Gimbal lock -ongelmasta.

Sensorialustaa liikutettiin ensin 90 akselin kulmaan x–akselin ympäri, jonka jälkeen käännettiin alustaa 180 astetta z–akselin ympäri. Tästä asennosta alustaa käännettiin 90 astetta y-akselin ympäri. Kaikki käännökset suoritettiin vaiheittain, eli käännöksiä tehtiin vain yhdelle akselille kerrallaan.



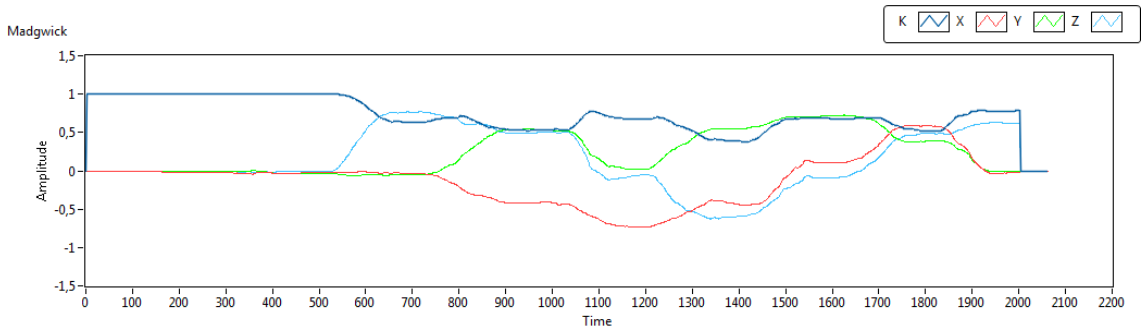

Kuvio 26. Raaka-arvonäytteiden arvojen kuvaaja kolmannesta liikkeestä.

Tämän kokeen raakanäytteet on esitetty kuviossa 26. Raakanäytteitä vertaamalla varsinaisiin suotimien tuottamiin arvoihin on kokeiden tulokset helppo toistaa. Kuvioissa 27 sekä 28 on kuvattuna näytteet suotimien läpi ajettuna. Kokeen tuottamien näytteiden analysoinnissa havaittiin huomattavia eroja kahden eri suotimen välillä. Koska havainnoidut erot olivat merkittävästi ennakko-odotuksia suurempia, toistettiin koe kahteen kertaan inhimillisten virheiden minimoimiseksi. Kummallakin koekerralla huomattiin suodinten käytöksessä sama ero.




Kuvio 27. Kalman-suotimen arvojen kuvaaja kolmannesta liikkeestä.

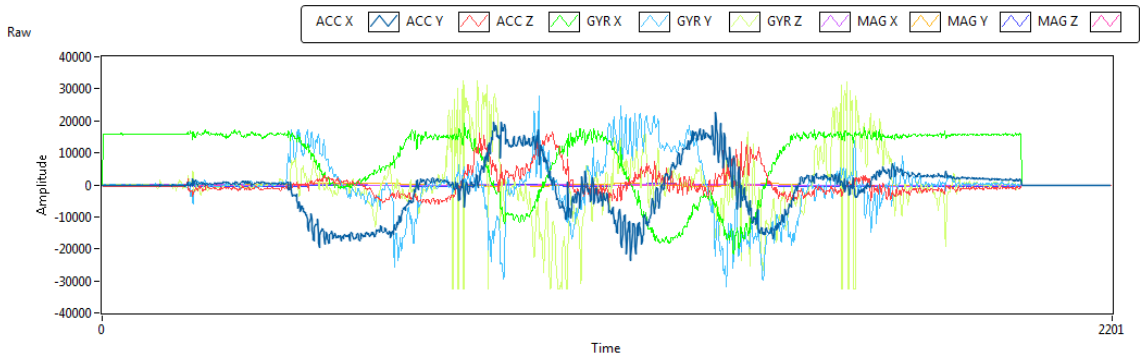
Kyseisessä kokeessa Kalman-suodin suoriutui huomattavasti heikommin verrattuna Madgwickin suotimeen. Vertaamalla keskenään kuvaajia 27 ja 28 erot käyvät ilmiselviksi. Kalman-suodin suoriutuu kokeesta hyvin ensimmäiseen 90 asteen käännökseen asti jonka voimme nähdä kuviossa 27 noin viidennensadannen näytteen kohdalla. Tämän jälkeen suoritetaan toinen 180 asteen käännös noin tuhannennen näytteen kohdalla. Näytteissä on havaittavissa suurta kohinaa sekä arvaamatonta käytöstä sensorialustan todellisen tilan kuvaamisessa. Suurin osa häiriöistä näytteissä on selitettävissä Gimbal lockilla, sillä akselien aiheuttama häiriö toisiin akseleihin on suurimmillaan juuri jonkin akselin ollessa 90 asteen kulmassa. Kuitenkin havaittu suuri kohina suotimessa oli odottamatonta.




Kuvio 28. Madgwickin suotimen arvojen kuvaaja kolmannesta liikkeestä.

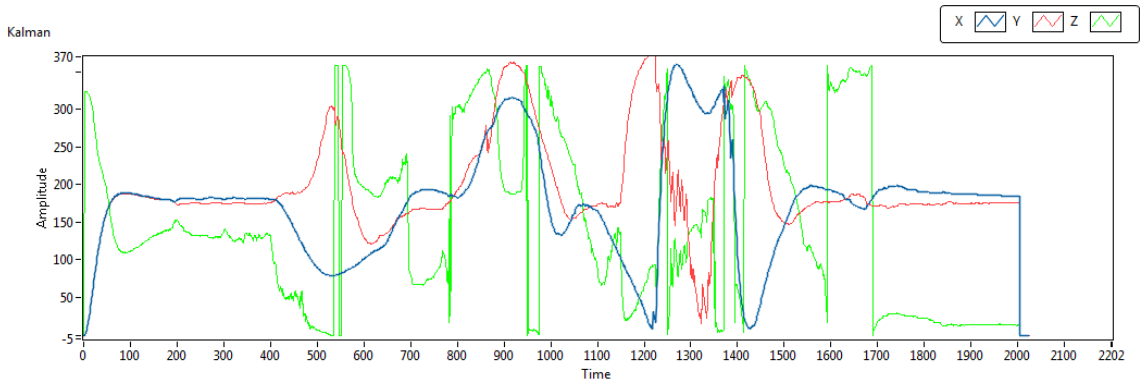
Kuviosta 28 pystymme havainnoimaan sen sijaan Madgwickin suotimen suoriutuneen kokeesta hyvin. Madgwickin suodin ei kärsi Gimbal lock -ongelmasta eikä arvoissa ole havaittavissa kohinaa.

Neljännessä kokeessa sensorialustaa kokeiltiin edelleen haastavamman liikesarjan tallentamisessa. Sensorialustaa liikutettiin 360 asteen käännöksiä sekä alustaa liikutettiin niin, että käännökset vaikuttivat useisiin eri akseleihin samanaikaisesti. Alustaa ei pidetty paikallaan neljännessä kokeessa, vaan se pidettiin kokoaikaisessa liikkeessä. Neljännen kokeen tuottamat näytteet on havainnollistettu kuvioissa 30 sekä 31. Raaka-arvot joista näytteet on johdettu, on kuvattu kuviossa 29. Kuviosta 29 voimme havainnoida liikesarjan aiheuttavan huomattavasti enemmän arvojen vaihtelua sensorien tuottamille arvoille, lisäten kohinaa ja näinollen sen olevan haastavampi näytekokonaisuus kummallekin suotimelle.



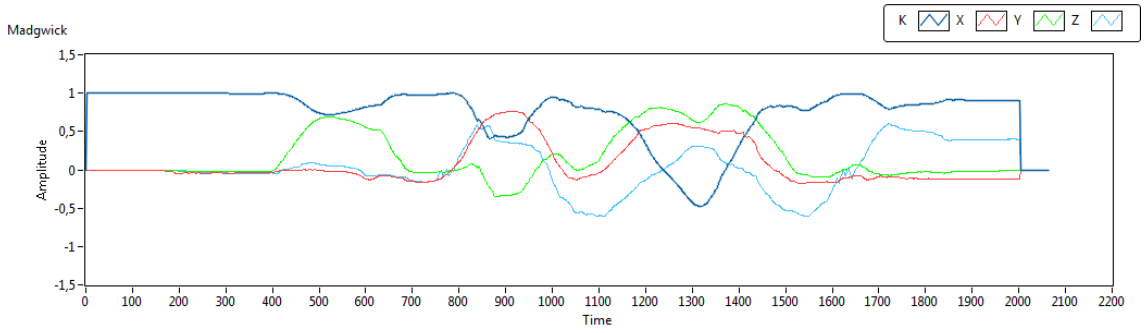

Kuvio 29. Raaka-arvonäytteiden arvojen kuvaaja neljännestä liikkeestä.

Suodinten suorituksissa oli myös neljännessä kokeessa huomattavissa huomattavia eroja. Kalman-suotimelle oli haasteellista suoriutua yli 360 asteen käännöksistä eikä suodin suoriutunut hyvin kahden eri akselin samanaikaisista käännöksistä. Suodin ei pystynyt havainnollistamaan sensorialustaan kohdistettuja todellisia rotaatioita riittävällä tarkkuudella. Kuviosta 30 pystymme havainnoimaan, että Kalman-suotimen tuottamissa arvoissa ei kuitenkaan esiinny kohinaa, tarkoittaen, että suodin on onnistunut suodattamaan raaka-näytteissä havaitun kohinan täysin pois. Kohinaa esiintyy ainoastaan tiettyinä hetkinä näytteissä, jolloin kaksi akselia on asettunut linjaan.




Kuvio 30. Kalman-suotimen arvojen kuvaaja neljännestä liikkeestä.

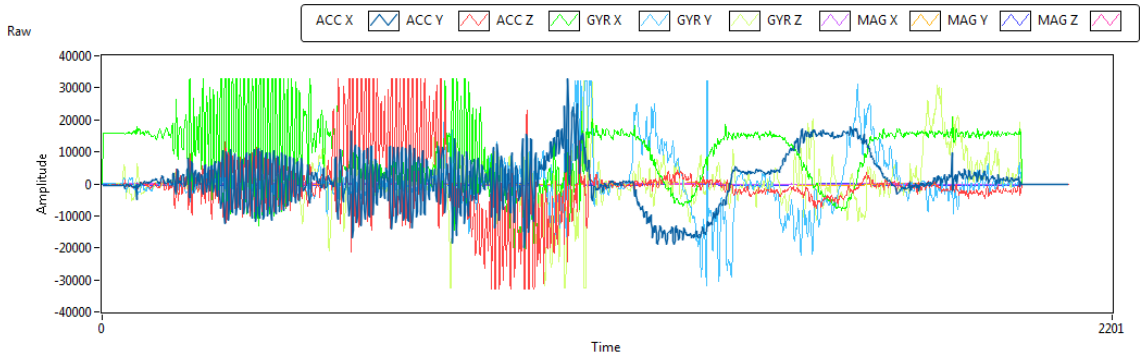
Myös Madgwickin suotimen käytöksessä oli havaittavissa häiriökäyttäytymisessä vaikkakin pienempää kuin Kalman-suotimen käytöksessä. Madgwickin suodin ei onnistunut kuvaamaan liian nopeasti toteutettuja rotaatioita riittävällä tarkkuudella. Suotimessa oli siis havaittavissa sisäistä viivettä. Sen sijaan muut rotaatiot Madgwickin suodin onnistui kuvaamaan täydellisesti. Tämä sisäinen viive on selitettävissä virheellä suotimen ajastuksessa, joka ei koetilanteessa ollut laskennallisessa optimiarvossaan.




Kuvio 31. Madgwickin suotimen arvojen kuvaaja neljännestä liikkeestä.

Kuviosta 31 pystymme havainnoimaan Madgwickin tuottamien näytteiden arvojen olevan vakaita. Suodin on siis onnistunut suodattamaan raaka-arvoissa havainnoidun kohinan pois.

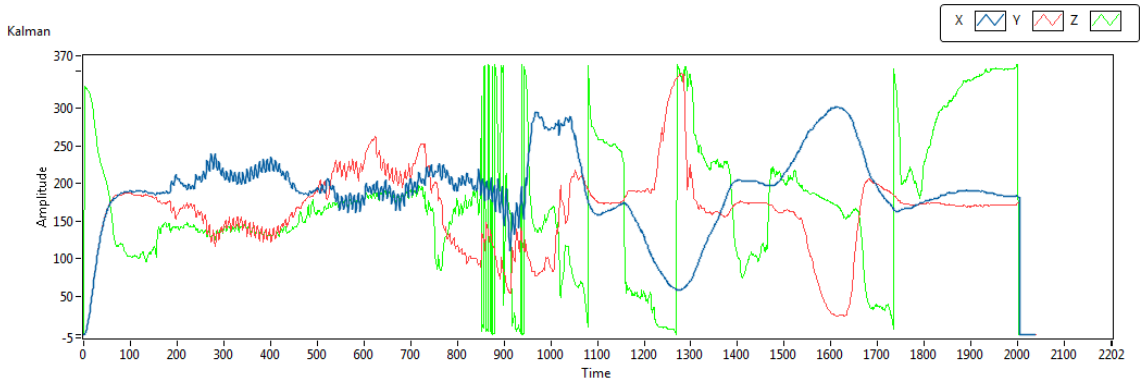
Viidennessä kokeessa liike pidettiin samanlaisena kuin neljännessä kokeessa, mutta koeympäristöön tuotiin manuaalisesti tuotettua tärinää. Viidennen kokeen odotettiin olevan suotimille ehdottomasti haasteellisin ja huomattavaa häiriökäyttäytymistä odotettiin esiintyvän. Tämän kokeen sensorialustan tuottamat raaka-arvot on esitetty kuviossa 32.




Kuvio 32. Raaka-arvonäytteiden arvojen kuvaaja viidennestä liikkeestä.

Kuviosta 32 pystytään havainnoimaan kohinaa esiintyvän erityisen paljon kiihtyvyysanturin tuottamissa arvoissa. Erityiseksi haasteeksi suotimille muodostuu viidennessä kokeessa oikeaan sensoriin luottaminen sekä havainnoidun kohinan suodattaminen pois kuitenkin sellaisella tavalla, ettei se vaikuta sensorialustan todellisen asennon kuvaamiseen negatiivisesti.

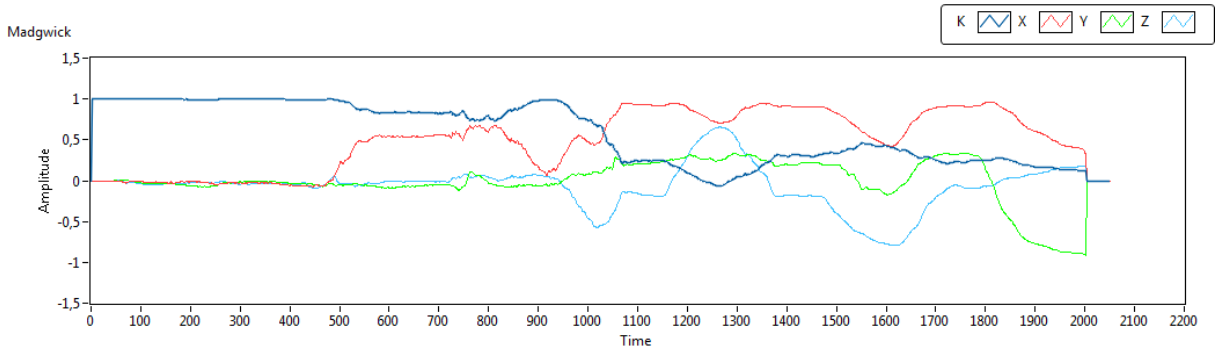
Kalman-suotimen tuottamat arvot on esitetty kuviossa 33. Arvoissa on havaittavissa huomattavaa kohinaa noin tuhannenteen näytepisteeseen asti. Tämän pisteen jälkeen kohinaa esiintyy vähemmän suotimen tuottamissa arvoissa. Voidaan päätellä, että suotimella kestää tämä aika sopeutua tärinäalttiiseen ympäristöön ja mukauttaa omat säätöterminsä sen mukaiseksi.




Kuvio 33. Kalman-suotimen arvojen kuvaaja viidennestä liikkeestä.

Tämän lisäksi Kalman-suodin ei onnistunut kuvaamaan sensorialustan todellista rotaatiota riittävällä tarkkuudella. Koeympäristöön tuotu tärinä vaikutti huomattavasti Kalman-suotimen kykyyn kuvata rotaatioita todenmukaisesti. Luultavasti optimoiduilla säätötermeillä Kalman-suodin pystyisi mukautumaan näytteissä olemassa olevaan tärinään huomattavasti nopeammin, sekä kuvaamaan rotaatioita paremmalla tarkkuudella.

Madgwickin suotimen tuottamissa arvoissa oli havaittavissa kohinaa, mutta kuitenkin pienemmällä mittakaavalla kuin Kalman-suotimessa. Madgwickin suotimen tuottamat arvot on esitetty kuviossa 34.




Kuvio 34. Madgwickin suotimen arvojen kuvaaja viidennestä liikkeestä.

Madgwickin suodin onnistui kuvaamaan sensorialustaan kohdistettuja todellisia rotaatioita suhteellisen hyvin. Koeympäristöön kohdistettu tärinä ei vaikuttanut suotimen suorituskykyyn huomattavasti ja suodin onnistui suodattamaan pois raaka-arvoissa esiintyneen tärinän tehokkaasti.

## 8.3 Johtopäätökset

Suoritetuista kokeista voidaan vetää johtopäätöksenä Madgwickin suodattimen toimivan tehokkaammin tässä dokumentissa kuvatussa applikaatiossa. Madgwickin suodin suoriutui paremmin kaikissa sensorialustaan kohdistetussa koeliikkeessä niin suodattamisen kuin todellisen rotaation kuvaamisen kannalta. Lisäksi alustavissa kokeissa todettiin Madgwickin suotimen soveltuvan paremmin mikrokontrollerialustoille. Madgwickin suodin on suunniteltu ja optimoitu käyttämään pelkästään sellaisia aritmeettisia laskutoimituksia, jotka on helppo toteuttaa muistirajoitetussa mikrokontrolleriympäristössä. Aritmeettisia operaatiota on Madgwickin suotimessa lähtökohtaisesti vähemmän, kuin Kalman-suotimessa. Kalman-suodin myöskin vaatii toteutukseensa hieman enemmän ohjelmointimuistia kuin tässä dokumentissa esitetty vastaparinsa. Mikäli tähän yhdistetään Kalman-suotimen ominainen esitystapa Eulerin kulmina, jotka esitysmuotona kärsivät ongelmista joista Madgwickin suotimen käyttämät kvaterniot eivät kärsi voidaan todeta Kalman-suotimen olevan heikompi vaihtoehto kuvatunlaisiin sovelluksiin.

Kalman-suodin on kuitenkin pitkäaikainen teollisuuden standardi, eikä täysin syyttä. Kalman-suotimen etuina on sen muuntautumiskyky erilaisiin tilanteisiin, jota Madgwickin suodin ei omaa, sekä olemassaoleva tietotaito Kalman-suotimista on suhteellisen uutta Madgwickin suodinta enempää. Huolellisella Kalman-suotimen optimoinnilla, pystyttäisiin suhteellisella varmuudella tuomaan Kalman-suotimen suorituskykyä lähemmäksi näissä kokeissa toteutetun Madgwickin suotimen tasoa.

Tiivistettynä Kalman-suodin on käyttökelpoinen sekä tehokas yleissuodin, joka omaa huolellisella optimoinnilla moneen tilanteeseen tärkeän muuntautumiskyvyn, mutta Madgwickin suodin toimii kuvantunlaisissa applikaatioissa tehokkaammin.

# 9 Jatkokehitys

Luvussa 8 esiteltyjen tutkimustulosten pohjalta todettiin tilanpaikannuspiirin kaipaavan jatkokehitystä. Jatkokehityksellä pyritään varmistamaan MARG-piirin toimintavarmuutta sekä laajentaa piirin mahdollisia käyttökohteita. Jatkokehityksen lopullisena tavoitteena on tuoda piirin toimintavarmuus tasolle, jossa sitä voitaisiin käyttää omavalmisteisen pitkän matkan lennokin lennonohjauspiirinä sekä stabilointiominaisuuksien parantaminen tasolle, jolla piiriä pystyttäisiin käyttämään em. lennokissa kameranohjausjärjestelmänä.

Jo valmistettu stabilointijärjestelmä voidaan tuoda lennokin lennonohjauspiiriksi suhteellisen pienillä muokkauksilla, siellä stabilointijärjestelmä pystytään muuntamaan servomoottoreiden ohjauksesta lennokin moottoreille sopivaksi ilman suurempia ohjelmistomuutoksia. Sen sijaan jatkokehityssuunnitelmassa pyritään keskittymään erityisesti suotimien toimintakyvyn parantamiseen, jotta se saadaan lennonohjausjärjestelmänä toimimisen vaatimalle tasolle.

Etenkin Kalman-suotimen suorituskyvyn varmistaminen on jatkokehityssuunnitelmassa korkealla prioriteetilla. Uskomme, että Kalman-suotimen toimintakykyä pystytään nostamaan suotimen säätötermien optimoinnilla. Ohjelmistoa pyritään jatkokehittämään mm. yksikkötestauksen muodossa. Kalman-suotimen jatkokehityksessä pyritään eliminoimaan mittausnäytteissä esiintynyttä kohinaa sekä takaamaan suotimen toimintavarmuus liikkeissä, jotka kääntävät piirin yli 90 asteen kulmaan.

Ohjelmiston jatkokehityssuunnitelmassa pyritään optimoimaan kummankin suotimen alustus- sekä kalibrointifunktiot. Alustavissa keskusteluissa projektiryhmän kesken on päätetty siirtyä suotimien alustamisessa sekä kalibroinnissa käytettyjen piirien ”self-test” funktioiden käyttöön. Näiden funktioiden käytön ohjelmassa uskotaan lisäävän piirin toimintavarmuutta sekä pienentävän piirin tarvitsemaa alustusaikaa, jolloin piirin tulee olla paikoillaan. Kalibrointialgoritmeissa pyritään myös ottamaan huomioon MEMS-piireissä esiintyvän lämpötilavaihtelun kompensointi.

Stabilointijärjestelmän jatkokehityksessä on päätetty muokata ohjelmistoa siten, että PID-säätimen säätötermejä pystytään muokkaamaan ohjelman ajon aikana. Säätötermeille pyritään myös hakemaan laskennalliset optimiarvot. Säätötermien säätäminen ohjelman ajon aikana takaa, että piiri pystyy mukautumaan kulloinkin kyseessä olevan applikaation erityistarpeisiin. Jotta stabilointijärjestelmän käyttöönottoa pystytään harkitsemaan toteutuskelpoisena lennonohjausjärjestelmänä, on säätötermit pystyttävä tuomaan ohjaimelle langattomasti, jolloin pystytään nopeasti reagoimaan stabilointiin vaikuttaviin ympäristömuuttujiin maasta käsin. Projektiryhmän keskusteluissa on harkittu käyttöönotettavaksi teknologiaksi mm. Bluetoothia.

Lisäksi keskusteluissa on myös kartoitettu uusien tilanpaikannustekniikoiden lisäämistä järjestelmään sekä olemassaolevien tilanpaikannusjärjestelmän ominaisuuksien laajentamista. Keskusteluissa esiintyneitä toteuttamiskelpoisia tekniikoita ovat olleet mm. GPS-piirin lisääminen järjestelmään.

# 10 Yhteenveto

Insinöörityön tarkoituksena oli toteuttaa kustannustehokas sekä käyttökelpoinen ratkaisu yleiskäyttöiseksi stabilointijärjestelmäksi sekä tutkia stabilointiin sekä tilanpaikannukseen käytettävien suotimien ominaisuuksia. Projekti aloitettiin ensin täysin käytäntömuotoisena, mutta se kasvoi lyhyessä ajassa laajuudeltaan myös tutkivaan suuntaan, kun stabilointijärjestelmän toteuttamiseksi alettiin tutkia syvällisemmin eri vaihtoehtoja.

Projektin alkaessa ei kummallakaan osallistujalla ollut syvällistä tietoa moderneista tilapaikannukseen käytettävistä metodeista. Voidaan todeta, että kumpikin projektin osanottajista oppi insinöörityön tekemisen aikana hyvin paljon nykyisestä tilanpaikannuksesta, mutta kattavaa tietoa saatiin myös kompleksista matematiikasta, ammattimaisesti toteutetusta tietotekniikan tutkimuksesta sekä projektinhallinnasta.

Kun vaihtoehdot oli rajattu kahteen, pitkäaikaiseen teollisuuden standardiin, Kalman-suotimeen sekä sen modernimpaan kilpailijaan, Madgwickin suotimeen, toteutettiin projektin käytännön osuus erittäin onnistuneesti. Projekti menestyi teknologiamessuilla järjestetyssä tekniikkakilpailussa sekä sitä harkittiin käyttöönottoon useaan eri ilmailualan projektiin.

Käytännön osuudessa toteutettiin onnistuneesti yleiskäyttöisen stabilointijärjestelmän prototyyppi kummallekin valitulle suotimelle, sekä toteutettiin onnistuneesti systemaattista jatkokehitystä kummallekin alustalle.

Insinöörityön tutkivassa osuudessa toteutettiin kattava vertailu valittujen suotimien suorituskyvystä sekä toteuttamiskelpoisuudesta muistirajatulle alustalle.

Projektin käytännön osuuteen oltiin tyytyväisiä, vaikkakin todettiin kokonaisuuden kaipaavan vielä jatkokehitystä toimiakseen toteuttamiskelpoisena vaihtoehtona nykymarkkinoilla kustannustehokkaana tilanpaikannusjärjestelmänä. Varsinkin toteutetun tutkimuksen tulosten varjossa on saatu käyttöön paljon uutta tietoa projektista, jonka ennustetaan auttavan projektin jatkokehityksessä huomattavasti.

Projektin aikana on saatu vertailukelpoista dataa kahdesta eri teollisuuden käyttämästä suotimesta. Varsinkin tämän tutkimusdatan myötä on projekti todettava erittäin onnistuneeksi.

## Lähteet


1 Madgwick, Sebastian. 2010. An efficient orientation filter for inertial and inertial/magnetic sensor arrays.

2 Jespersen Thomas. 2012. A practical approach to Kalman filter and how to implement it. < http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/>

3 Baker, Martin John. 2015. Maths - Euler Angles. <http://www.euclideanspace.com/maths/geometry/rotations/euler/>

4 Baker, Martin John. 2015. Maths - Quaternions. <http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/>

5 Kim, Phil. 2013. Rigid Body Dynamics For Beginners: Euler angles & Quaternions.

6 Paul Zarchan. 2009. Fundamentals of Kalman Filtering: A Practical Approach (Progress in Astronautics and Aeronautics).

7 Curtis, Howard. 2013. Orbital Mechanics for Engineering Students (Aerospace Engineering).

8 Fjeld, Paul & Jones, Eric. 2011. Gimbal Angles, Gimbal Lock, and a Fourth Gimbal for Christmas. <https://www.hq.nasa.gov/alsj/gimbals.html>

9 Unity Script Reference Guide. 2014. <http://docs.unity3d.com/ScriptReference/Quaternion.html>

10 Mahony Robert, Hamel, Tarek & Pflimlin, Jean Michael. 2008. Nonlinear Complementary Filters on the Special Orthogonal Group.

11 Pieter-Jan Van de Maele. 2013. Reading an IMU without Kalman: The Complementary Filter. <http://www.pieter-jan.com/node/11>

12 Кручу-верчу, запутать хочу: углы Эйлера и Gimbal lock. <http://habrahabr.ru/post/183116/>

13 Flight Instruments. <http://stickyfish.dk/Flight%20Instruments.html>

14 Icady. 2009. PID process control, a “Cruise Control” example. <http://www.codeproject.com/Articles/36459/PID-process-control-a-Cruise-Control-example>

15 National Instruments. 2012. PID Control. <http://www.ni.com/white-paper/6440/en/>

16 Araki, M. PID Controls.
<http://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf>

17 ATMEL, AVR Microcontrollers. 2013. Inter-Integrated Circuits – I2C Basics<http://maxembedded.com/2014/02/09/inter-integrated-circuits-i2c-basics/>

18 Philips. $I^2C$ standards. <http://www.i2c-bus.org/>

19 Tietotekniikan ja Elektroniikan seura. 2013 Verkkotiedote.
<http://eis.fi/1444/warkkaajat-tasavakisia-teknologia13-messuilla/>

20 Nyquist, H. 2002. Certain topics in telegraph transmission theory

21 Jaw-Kuen Shiau, Chen-Xuan Huang ja Ming-Yu Chang. 2012. Noise Characteristics of MEMS Gyro's Null Drift and Temperature Compensation.
<http://www2.tku.edu.tw/~tkjse/15-3/04-AE10102.pdf>

22 Erik B. Dam, Martin Koch & Martin Lillholm. 1998. Quaternions, interpolation and Animation <http://web.mit.edu/2.998/www/QuaternionReport1.pdf>

23 Cheng, Steve. 2013. Proof of Rodrigues' rotation formula
<http://planetmath.org/sites/default/files/texpdf/37221.pdf>

24 Bitsquid. 2013. What is gimbal lock and why do we still have to worry about it?
http://bitsquid.blogspot.fi/2013/03/what-is-gimbal-lock-and-why-do-we-still.html

## Lähdekoodi

Main.c

```
/* PID DEFINITIONS */
#define KP      4.3f
#define KI      0.02f
#define KD      4.5f

#include <Global.h>
#include <Initialization.h>
#include <calculate_angles.h>

/*   ISR MOFO   */
uint8 timerVariable = 0;

/*                    STRUCT LINKERS                    */
sensor gyro;
sensor acc;
sensor magnet;
sensor gyro_offset;
sensor acc_offset;

Angles imu_angles;
Angles magnet_angles;
Angles gyro_rates;
Angles kalmanized_angles;

/*         EXT FUNCTION PROTOTYPES          */
extern void tilt_calc(sensor* acc_bits, Angles* Angles);
extern void compensateTilt(sensor *magnet_raw, Angles *acc);

/*      FÄÄNKSHÖNS          */
float PID (int setpoint, float angle, float *error, float *prevError,
float *integ);
CY_ISR_PROTO(interruptHandler);

/*                    GLOBL MOFOS                                    */
    uint8 timerFag = 0;

/*      PID VARIABLES       */
float error_roll = 0, prevError_roll = 0, integ_roll = 0;
float error_pitch = 0, prevError_pitch = 0, integ_pitch = 0;
float error_yaw = 0, prevError_yaw = 0, integ_yaw = 0;

int main(void)
{


           char cBuffer[20];
           uint8 readBuffer[14];
    int16 iBuffer;
    float PID_Kulma_Roll = 0;
    float PID_Kulma_Pitch = 0;
    //PID VARIABLES

           //DELAYT JOTTA XBEE EHTII OTTAA YHTEYDEN VERKKOOON
```

```
            CyDelay(3000);

            sysInit();
    PWM_isr_StartEx(interruptHandler);
            PWM_isr_Enable();
    PWM_Enable();
            while(1)
            {

//         while(timerVariable == 0);
//         timerVariable=0;

                        HMC5883l();


                        mpu6050_read_register((uint8 *)readBuffer,
mpu6050_reg_accel_x_msb, 14u);

        acc.x = ((readBuffer[0]<<8)|readBuffer[1]);
                        acc.y = ((readBuffer[2]<<8)|readBuffer[3]);
                        acc.z = ((readBuffer[4]<<8)|readBuffer[5]);

                        //mpu6050_read_register((uint8 *)readBuffer,
mpu6050_reg_gyro_x_msb, 6u);

           gyro.x = ((readBuffer[8]<<8)|readBuffer[9]);
                        gyro.y = ((readBuffer[10]<<8)|readBuffer[11]);
                        gyro.z = ((readBuffer[12]<<8)|readBuffer[13]);

        imu_angles.pitch = (float)(acc.x/ACC_SCALE_FACTOR);
        imu_angles.roll = (float)(acc.y/ACC_SCALE_FACTOR);
        imu_angles.yaw = (float)(acc.z/ACC_SCALE_FACTOR);

        gyro_rates.pitch =
(((float)(gyro.x)/(float)(GYRO_SCALE_FACTOR)));
                        gyro_rates.roll =
(((float)(gyro.y)/(float)(GYRO_SCALE_FACTOR)));
        gyro_rates.yaw =
(((float)(gyro.z)/(float)(GYRO_SCALE_FACTOR)));

        magnet_angles.pitch =
(float)((float)(magnet.x)*MAG_SCALE_FACTOR);
        magnet_angles.roll =
(float)((float)(magnet.y)*MAG_SCALE_FACTOR);
        magnet_angles.yaw =
(float)((float)(magnet.z)*MAG_SCALE_FACTOR);

        tilt_calc(&acc, &imu_angles);
//calculate tilt for accelerometer
                        compensateTilt(&magnet, &imu_angles);
//tilt compensation, duh.
                        tilt_calc(&magnet, &magnet_angles);
                //calculate yaw with magnetometer, using compensated
values

                        /* CONVERT FROM RADIANS TO DEGREES */
                        imu_angles.roll = RADTODEG*(imu_angles.roll);
                        imu_angles.pitch = RADTODEG*(imu_angles.pitch);
```

```
                        magnet_angles.yaw =
rad_to_deg(magnet_angles.yaw);

                        /* RELEASE THE KALMAN! */
                        kalmanized_angles.pitch =
kalman_filter(imu_angles.pitch, gyro_rates.pitch
,&kalman_variables_pitch);
                        kalmanized_angles.roll =
kalman_filter(imu_angles.roll, gyro_rates.roll
,&kalman_variables_roll);
                        //kalmanized_angles.yaw =
kalman_filter(magnet_angles.yaw, gyro_rates.yaw,
&kalman_variables_yaw);

        iBuffer = (int16)(kalmanized_angles.pitch);
        sprintf(cBuffer, "%d", iBuffer);
        UART_PutString(cBuffer);
                        UART_PutString(" ");

        iBuffer = (int16)(kalmanized_angles.roll);
        sprintf(cBuffer, "%d", iBuffer);
                        UART_PutString(cBuffer);
                        UART_PutString(" ");

        iBuffer = (int16)(magnet_angles.yaw);
        sprintf(cBuffer, "%d", iBuffer);
                        UART_PutString(cBuffer);
                        UART_PutString(" ");

        iBuffer = (int16)(PID_Kulma_Roll);
        sprintf(cBuffer, "%d", iBuffer);
                        UART_PutString(cBuffer);
                        UART_PutString(" ");

        iBuffer = (int16)(PID_Kulma_Pitch);
        sprintf(cBuffer, "%d", iBuffer);
                        UART_PutString(cBuffer);
                        UART_PutString(" ");

        UART_PutCRLF(0);

        PID_Kulma_Roll = PID(180, kalmanized_angles.roll, &error_roll,
&prevError_roll, &integ_roll);
        PID_Kulma_Pitch = PID(180, kalmanized_angles.pitch,
&error_pitch, &prevError_pitch, &integ_pitch);

        PWM_WriteCompare1(1999 + PID_Kulma_Roll);
        PWM_WriteCompare2(1999 + PID_Kulma_Pitch);
           }

    return 0;
}

float PID (int setpoint, float angle, float *error, float *prevError,
float *integ)
{
    float output;
    float deri;
```

```
    *error = setpoint - angle;

    *integ += *error;

    deri = (*error - *prevError);

    output = (KP * *error) + (KI * *integ) + (KD * deri);

    *prevError = *error;

    return (output);
}
CY_ISR(interruptHandler)
{
    timerVariable=1;
    timerFag ^= 1;
    Timer_Check_Write(timerFag);
    PWM_ClearFIFO();
    PWM_STATUS;

}
```

MPU6050.c

```
#include <mpu6050.h>

int mpu6050_Start(void) {
    CyDelay(500);

    /*Reset to defaults*/
    //mpu6050_write_register(mpu6050_reg_pwr_mgmt_1,0x80u);

    /*Write default config*/
    mpu6050_write_register(mpu6050_reg_pwr_mgmt_1,0x03);          //
POWER MANAGMENT, VALITAAN KELLOKSI GYRON ZAXIS
          mpu6050_write_register(mpu6050_reg_smprt_div,8u);      //
GYRON OUTPUT RATE, LSKETAAN KAAVALLA: kun dlp_cfg=0, 8kHZ /
(rekisteriin kirjoitettu arvo (7) +1)
//    mpu6050_write_register(mpu6050_reg_pwr_mgmt_2,0u);
          mpu6050_write_register(mpu6050_reg_config, 6u);
    mpu6050_write_register(mpu6050_reg_gyro_config, 8u);
    mpu6050_write_register(mpu6050_reg_accel_config, 16u);

    UART_PutString("MPU6050 INIT OK...\r\n");
          UART_PutCRLF(0);

    return 0u;
}

int mpu6050_read_register(unsigned char *readData, unsigned char
reg_address, unsigned char read_count) {



I2C_I2CMasterWriteBuf(MPU6050_ADDRESS,&reg_address,1,I2C_I2C_MODE_COMP
LETE_XFER);

    while(0u == (I2C_I2CMasterStatus() & I2C_I2C_MSTAT_WR_CMPLT));
    I2C_I2CMasterClearStatus();


I2C_I2CMasterReadBuf(MPU6050_ADDRESS,readData,read_count,I2C_I2C_MODE_
COMPLETE_XFER);

    while(0u == (I2C_I2CMasterStatus() & I2C_I2C_MSTAT_RD_CMPLT));
    I2C_I2CMasterClearStatus();

    return 0u;
}

int mpu6050_write_register(unsigned char reg_address, unsigned char
writeData) {


    int i, status;
    unsigned char buffer[2];
    buffer[0] = reg_address;
    buffer[1] = writeData;
```

```
    status = I2C_I2CMasterSendStart(MPU6050_ADDRESS,
I2C_I2C_WRITE_XFER_MODE);

    if(I2C_I2C_MSTR_NO_ERROR == status) /* Check if transfer completed
without errors */
    {
        for(i = 0; i < 2; i++)
        {
            status = I2C_I2CMasterWriteByte(buffer[i]);
            if(I2C_I2C_MSTR_NO_ERROR != status)
            {
            break;
            }
        }
    }
    I2C_I2CMasterSendStop(); /* Send Stop */

//    LCD_Char_PrintInt8(reg_address);
//    LCD_Char_PrintInt8(writeData);
//
//    I2C_I2CMasterClearStatus();
//
//
I2C_I2CMasterWriteBuf(MPU6050_ADDRESS,buffer,2u,I2C_I2C_MODE_COMPLETE_
XFER);
//    while(0u == (I2C_I2CMasterStatus() & I2C_I2C_MSTAT_WR_CMPLT));
//    I2C_I2CMasterClearStatus();
//
//
    return 0u;
}
/* [] END OF FILE */
```

Kalman.c

```
/* INIT P AND Q VALUES */
#include <Kalman_Filter.h>

kalmanVariables kalman_variables_pitch;
kalmanVariables kalman_variables_roll;
kalmanVariables kalman_variables_yaw;

void initKalman(kalmanVariables *init_this)
{
            init_this->Q_angle = 0.005;
            init_this->Q_gyroBias = 0.03;
            init_this->R_angle = 0.02;
            init_this->dt = 0.05;

            init_this->bias = 0;              // Gyron driftaus
            init_this->angle = 0;             // angle on filtter[illegible]it[illegible]v[illegible]
arvo
            init_this->P[0][0] = 0;                       //
            init_this->P[0][1] = 0;
            init_this->P[1][0] = 0;
            init_this->P[1][1] = 0;
            init_this->K[0] = 0;              // Kalman Gain factor
            init_this->K[1] = 0;

}

float kalman_filter(float newAngle, float newRate, kalmanVariables
*kVariables) //newAngle kiihtyvyysanturin antama arvo, newRate
Gyroskoopin antama arvo
{
            float S;
            // Luotettavuuskerroin
            float y;
            // Mittaus miinus arvio mittauksesta
            float rate;
            // rate = Todellinen mittausarvo


            rate = newRate - kVariables->bias;
            // newRate = Gyron antama arvo
            kVariables->angle += kVariables->dt * rate;
                        //

            kVariables->P[0][0] += kVariables->dt * (kVariables-
>dt*kVariables->P[1][1] - kVariables->P[0][1] - kVariables->P[1][0] +
kVariables->Q_angle); // p[illegible]ivitet[illegible]n todenn[illegible]k[illegible]isyysmatriisi
            kVariables->P[0][1] -= kVariables->dt * kVariables-
>P[1][1];
            kVariables->P[1][0] -= kVariables->dt * kVariables-
>P[1][1];
            kVariables->P[1][1] += kVariables->Q_gyroBias * kVariables-
>dt;

            y = newAngle - kVariables->angle;          // ACTUAL sensor
data - Prediction of sensor data
```

```
            S = kVariables->P[0][0] + kVariables->R_angle; // joka
kierroksella p�ivitet��n luotettavuuskerroin kovarianssin ja
edellisten mittausten suhteen

            kVariables->K[0] = kVariables->P[0][0] / S;
            //suhteutetaan nykyinen mittausarvo luotettavuuskertoimeen
            kVariables->K[1] = kVariables->P[1][0] / S;

            kVariables->angle += kVariables->K[0] * y;            //
p�ivitet��n kulma mittaustulosten ja Sensoridatan ja sensoridatan
arvion suhteen
            kVariables->bias += kVariables->K[1] * y;             //
p�ivitet��n gyron driftaus

            kVariables->P[0][0] -= kVariables->K[0] * kVariables-
>P[0][0]; //p�ivitet��n todenn�k�isyysmatriisi, koska virhearvio on
pienentynyt
            kVariables->P[0][1] -= kVariables->K[0] * kVariables-
>P[0][1];
            kVariables->P[1][0] -= kVariables->K[1] * kVariables-
>P[0][0];
            kVariables->P[1][1] -= kVariables->K[1] * kVariables-
>P[0][1];

            return (kVariables->angle);
}
```

mahony.c

```
#include "mahony.h"
#include <math.h>

//---------------------------------------------------------------------
//------------------------------
// Definitions

#define sampleFreq  170.0f//170.0f                    // sample fre-
quency in Hz
#define betaDef           0.1f              // 2 * proportional gain

//---------------------------------------------------------------------
//------------------------------
// Variable definitions

volatile float beta = betaDef;
                                          // 2 * proportional gain
(Kp)
volatile float q0 = 1.0f, q1 = 0.0f, q2 = 0.0f, q3 = 0.0f;       //
quaternion of sensor frame relative to auxiliary frame

//---------------------------------------------------------------------
//------------------------------
// Function declarations

float invSqrt(float x);

//=====================================================================
==============================
// Functions

//---------------------------------------------------------------------
//------------------------------
// AHRS algorithm update

void MadgwickAHRSupdate(float gx, float gy, float gz, float ax, float
ay, float az, float mx, float my, float mz) {
          float recipNorm;
          float s0, s1, s2, s3;
          float qDot1, qDot2, qDot3, qDot4;
          float hx, hy;
          float _2q0mx, _2q0my, _2q0mz, _2q1mx, _2bx, _2bz, _4bx,
_4bz, _2q0, _2q1, _2q2, _2q3, _2q0q2, _2q2q3, q0q0, q0q1, q0q2, q0q3,
q1q1, q1q2, q1q3, q2q2, q2q3, q3q3;

          // Use IMU algorithm if magnetometer measurement invalid
(avoids NaN in magnetometer normalisation)
          if((mx == 0.0f) && (my == 0.0f) && (mz == 0.0f)) {
                    MadgwickAHRSupdateIMU(gx, gy, gz, ax, ay, az);
                    return;
          }

          // Rate of change of quaternion from gyroscope
          qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);
          qDot2 = 0.5f * (q0 * gx + q2 * gz - q3 * gy);
          qDot3 = 0.5f * (q0 * gy - q1 * gz + q3 * gx);
          qDot4 = 0.5f * (q0 * gz + q1 * gy - q2 * gx);
```

```
		// Compute feedback only if accelerometer measurement valid
(avoids NaN in accelerometer normalisation)
		if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f))) {

			// Normalise accelerometer measurement
			recipNorm = invSqrt(ax * ax + ay * ay + az * az);
			ax *= recipNorm;
			ay *= recipNorm;
			az *= recipNorm;

			// Normalise magnetometer measurement
			recipNorm = invSqrt(mx * mx + my * my + mz * mz);
			mx *= recipNorm;
			my *= recipNorm;
			mz *= recipNorm;

			// Auxiliary variables to avoid repeated arithme-
tic
			_2q0mx = 2.0f * q0 * mx;
			_2q0my = 2.0f * q0 * my;
			_2q0mz = 2.0f * q0 * mz;
			_2q1mx = 2.0f * q1 * mx;
			_2q0 = 2.0f * q0;
			_2q1 = 2.0f * q1;
			_2q2 = 2.0f * q2;
			_2q3 = 2.0f * q3;
			_2q0q2 = 2.0f * q0 * q2;
			_2q2q3 = 2.0f * q2 * q3;
			q0q0 = q0 * q0;
			q0q1 = q0 * q1;
			q0q2 = q0 * q2;
			q0q3 = q0 * q3;
			q1q1 = q1 * q1;
			q1q2 = q1 * q2;
			q1q3 = q1 * q3;
			q2q2 = q2 * q2;
			q2q3 = q2 * q3;
			q3q3 = q3 * q3;

			// Reference direction of Earth's magnetic field
			hx = mx * q0q0 - _2q0my * q3 + _2q0mz * q2 + mx *
q1q1 + _2q1 * my * q2 + _2q1 * mz * q3 - mx * q2q2 - mx * q3q3;
			hy = _2q0mx * q3 + my * q0q0 - _2q0mz * q1 +
_2q1mx * q2 - my * q1q1 + my * q2q2 + _2q2 * mz * q3 - my * q3q3;
			_2bx = sqrt(hx * hx + hy * hy);
			_2bz = -_2q0mx * q2 + _2q0my * q1 + mz * q0q0 +
_2q1mx * q3 - mz * q1q1 + _2q2 * my * q3 - mz * q2q2 + mz * q3q3;
			_4bx = 2.0f * _2bx;
			_4bz = 2.0f * _2bz;

			// Gradient decent algorithm corrective step
			s0 = -_2q2 * (2.0f * q1q3 - _2q0q2 - ax) + _2q1 *
(2.0f * q0q1 + _2q2q3 - ay) - _2bz * q2 * (_2bx * (0.5f - q2q2 - q3q3)
+ _2bz * (q1q3 - q0q2) - mx) + (-_2bx * q3 + _2bz * q1) * (_2bx *
(q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + _2bx * q2 * (_2bx * (q0q2
+ q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
			s1 = _2q3 * (2.0f * q1q3 - _2q0q2 - ax) + _2q0 *
(2.0f * q0q1 + _2q2q3 - ay) - 4.0f * q1 * (1 - 2.0f * q1q1 - 2.0f *
```

```
q2q2 - az) + _2bz * q3 * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 -
q0q2) - mx) + (_2bx * q2 + _2bz * q0) * (_2bx * (q1q2 - q0q3) + _2bz *
(q0q1 + q2q3) - my) + (_2bx * q3 - _4bz * q1) * (_2bx * (q0q2 + q1q3)
+ _2bz * (0.5f - q1q1 - q2q2) - mz);
                    s2 = -_2q0 * (2.0f * q1q3 - _2q0q2 - ax) + _2q3 *
(2.0f * q0q1 + _2q2q3 - ay) - 4.0f * q2 * (1 - 2.0f * q1q1 - 2.0f *
q2q2 - az) + (-_4bx * q2 - _2bz * q0) * (_2bx * (0.5f - q2q2 - q3q3) +
_2bz * (q1q3 - q0q2) - mx) + (_2bx * q1 + _2bz * q3) * (_2bx * (q1q2 -
q0q3) + _2bz * (q0q1 + q2q3) - my) + (_2bx * q0 - _4bz * q2) * (_2bx *
(q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
                    s3 = _2q1 * (2.0f * q1q3 - _2q0q2 - ax) + _2q2 *
(2.0f * q0q1 + _2q2q3 - ay) + (-_4bx * q3 + _2bz * q1) * (_2bx * (0.5f
- q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (-_2bx * q0 + _2bz * q2)
* (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + _2bx * q1 *
(_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
                    recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 +
s3 * s3); // normalise step magnitude
                    s0 *= recipNorm;
                    s1 *= recipNorm;
                    s2 *= recipNorm;
                    s3 *= recipNorm;

                    // Apply feedback step
                    qDot1 -= beta * s0;
                    qDot2 -= beta * s1;
                    qDot3 -= beta * s2;
                    qDot4 -= beta * s3;
          }

          // Integrate rate of change of quaternion to yield quater-
nion
          q0 += qDot1 * (1.0f / sampleFreq);
          q1 += qDot2 * (1.0f / sampleFreq);
          q2 += qDot3 * (1.0f / sampleFreq);
          q3 += qDot4 * (1.0f / sampleFreq);

          // Normalise quaternion
          recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
          q0 *= recipNorm;
          q1 *= recipNorm;
          q2 *= recipNorm;
          q3 *= recipNorm;

    q.w = q0;
    q.x = q1;
    q.y = q2;
    q.z = q3;
}

//---------------------------------------------------------------------
-------------------------------
// IMU algorithm update

void MadgwickAHRSupdateIMU(float gx, float gy, float gz, float ax,
float ay, float az) {
          float recipNorm;
          float s0, s1, s2, s3;
          float qDot1, qDot2, qDot3, qDot4;
```

```
		float _2q0, _2q1, _2q2, _2q3, _4q0, _4q1, _4q2 ,_8q1, _8q2,
q0q0, q1q1, q2q2, q3q3;

		// Rate of change of quaternion from gyroscope
		qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);
		qDot2 = 0.5f * (q0 * gx + q2 * gz - q3 * gy);
		qDot3 = 0.5f * (q0 * gy - q1 * gz + q3 * gx);
		qDot4 = 0.5f * (q0 * gz + q1 * gy - q2 * gx);

		// Compute feedback only if accelerometer measurement valid
(avoids NaN in accelerometer normalisation)
		if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f))) {

			// Normalise accelerometer measurement
			recipNorm = invSqrt(ax * ax + ay * ay + az * az);
			ax *= recipNorm;
			ay *= recipNorm;
			az *= recipNorm;

			// Auxiliary variables to avoid repeated arithme-
tic
			_2q0 = 2.0f * q0;
			_2q1 = 2.0f * q1;
			_2q2 = 2.0f * q2;
			_2q3 = 2.0f * q3;
			_4q0 = 4.0f * q0;
			_4q1 = 4.0f * q1;
			_4q2 = 4.0f * q2;
			_8q1 = 8.0f * q1;
			_8q2 = 8.0f * q2;
			q0q0 = q0 * q0;
			q1q1 = q1 * q1;
			q2q2 = q2 * q2;
			q3q3 = q3 * q3;

			// Gradient decent algorithm corrective step
			s0 = _4q0 * q2q2 + _2q2 * ax + _4q0 * q1q1 - _2q1
* ay;
			s1 = _4q1 * q3q3 - _2q3 * ax + 4.0f * q0q0 * q1 -
_2q0 * ay - _4q1 + _8q1 * q1q1 + _8q1 * q2q2 + _4q1 * az;
			s2 = 4.0f * q0q0 * q2 + _2q0 * ax + _4q2 * q3q3 -
_2q3 * ay - _4q2 + _8q2 * q1q1 + _8q2 * q2q2 + _4q2 * az;
			s3 = 4.0f * q1q1 * q3 - _2q1 * ax + 4.0f * q2q2 *
q3 - _2q2 * ay;
			recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 +
s3 * s3); // normalise step magnitude
			s0 *= recipNorm;
			s1 *= recipNorm;
			s2 *= recipNorm;
			s3 *= recipNorm;

			// Apply feedback step
			qDot1 -= beta * s0;
			qDot2 -= beta * s1;
			qDot3 -= beta * s2;
			qDot4 -= beta * s3;
		}
```

```
            // Integrate rate of change of quaternion to yield quater-
nion
            q0 += qDot1 * (1.0f / sampleFreq);
            q1 += qDot2 * (1.0f / sampleFreq);
            q2 += qDot3 * (1.0f / sampleFreq);
            q3 += qDot4 * (1.0f / sampleFreq);

            // Normalise quaternion
            recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
            q0 *= recipNorm;
            q1 *= recipNorm;
            q2 *= recipNorm;
            q3 *= recipNorm;

    q.w = q0;
    q.x = q1;
    q.y = q2;
    q.z = q3;

}

//---------------------------------------------------------------------
---------------------------------
// Fast inverse square-root
// See: http://en.wikipedia.org/wiki/Fast_inverse_square_root

float invSqrt(float x) {
            float halfx = 0.5f * x;
            float y = x;
            long i = *(long*)&y;
            i = 0x5f3759df - (i>>1);
            y = *(float*)&i;
            y = y * (1.5f - (halfx * y * y));
            return y;
}
```

Initialization.c

```
#include <Initialization.h>

void sysInit(void)
{
            CyGlobalIntEnable;

    Clock_1_Enable();
    Clock_1_Start();

    PWM_Start();
    PWM_WriteCompare1(0);


            UART_Start();          // Enable UART
            UART_PutString("UART INIT OK....\r\n");

            I2C_Start();
            UART_PutString("I2C INIT OK...\r\n");

            init_sensor_values(&acc);
            init_sensor_values(&gyro);

            init_sensor_values(&acc_offset);
            init_sensor_values(&gyro_offset);

            init_imu_angle_values(&imu_angles);
            init_imu_angle_values(&gyro_rates);
            init_imu_angle_values(&kalmanized_angles);

            initKalman(&kalman_variables_pitch);
            initKalman(&kalman_variables_roll);
            initKalman(&kalman_variables_yaw);

            mpu6050_Start();
            init_HMC5883l();
    calibrate_sensors();

}

void init_sensor_values(sensor *init_this)
{
            init_this->x = 0;
            init_this->y = 0;
            init_this->z = 0;
}
void init_imu_angle_values(Angles *init_this)
{
            init_this->yaw = 0;
            init_this->pitch = 0;
            init_this->roll = 0;
}

void calibrate_sensors(void)
{
            unsigned char rxBuffer[6] = {0};
            int32 sensor_cal_x_temp = 0;
            int32 sensor_cal_y_temp = 0;
```

```
            int32 sensor_cal_z_temp = 0;
            int loop_count = 0;


            for(loop_count=0 ; loop_count < 128 ; loop_count++)
            {
                        mpu6050_read_register((uint8 *)rxBuffer,
mpu6050_reg_accel_x_msb, 6u);

                        sensor_cal_x_temp +=
((rxBuffer[0]<<8)|rxBuffer[1]);
                        sensor_cal_y_temp +=
((rxBuffer[2]<<8)|rxBuffer[3]);
                        sensor_cal_z_temp +=
((rxBuffer[4]<<8)|rxBuffer[5]);

            }

            acc_offset.x = (int16)(sensor_cal_x_temp / 128);
            acc_offset.y = (int16)(sensor_cal_y_temp / 128);
            acc_offset.z = (int16)(sensor_cal_z_temp / 128);

            sensor_cal_x_temp = 0;
            sensor_cal_y_temp = 0;
            sensor_cal_z_temp = 0;

            for(loop_count=0 ; loop_count<128 ; loop_count++)
            {

                mpu6050_read_register((uint8 *)rxBuffer,
mpu6050_reg_gyro_x_msb, 6u);

        sensor_cal_x_temp += ((rxBuffer[0]<<8)|rxBuffer[1]);
                        sensor_cal_y_temp +=
((rxBuffer[2]<<8)|rxBuffer[3]);
                        sensor_cal_z_temp +=
((rxBuffer[4]<<8)|rxBuffer[5]);

            }

            gyro_offset.x = (int16)(sensor_cal_x_temp / 128);
            gyro_offset.y = (int16)(sensor_cal_y_temp / 128);
            gyro_offset.z = (int16)(sensor_cal_z_temp / 128);

            UART_PutString("Calibration INIT OK....\r\n");
}
```

HMC5883l.c

```
#include <HMC5883l.h>

#define BYTE                                    uint8
#define HMC_REG                 0x1E

void HMC5883l (void)
{
            BYTE bStatus = 0, i;
            BYTE bMagnetData[6] = {0};

            bStatus = (I2C_I2CMasterSendStart(HMC_REG,
I2C_I2C_WRITE_XFER_MODE));
            if (I2C_I2C_MSTR_NO_ERROR == bStatus)
            {
                        I2C_I2CMasterWriteByte(0x03);
                        I2C_I2CMasterSendRestart(HMC_REG,
I2C_I2C_READ_XFER_MODE);

                        for (i=0;i<5;i++)
                        {
                                    bMagnetData[i] =
I2C_I2CMasterReadByte(I2C_I2C_ACK_DATA);
                        }

                        I2C_I2CMasterReadByte(I2C_I2C_NAK_DATA);

                        I2C_I2CMasterSendStop();

                        magnet.x = (bMagnetData[0]<<8) | bMagnetData[1];
                        magnet.z = (bMagnetData[2]<<8) | bMagnetData[3];
                        magnet.y = (bMagnetData[4]<<8) | bMagnetData[5];

            }

            else
            {
                        UART_PutCRLF(0);
                        UART_PutString("c=====3");
                        UART_PutCRLF(0);
            }

}

BYTE init_HMC5883l (void)
{
            BYTE bStatus = 0;

            bStatus = (I2C_I2CMasterSendStart(HMC_REG,
I2C_I2C_WRITE_XFER_MODE));

            if (I2C_I2C_MSTR_NO_ERROR == bStatus)
            {
                        I2C_I2CMasterWriteByte(0x00);
                        I2C_I2CMasterWriteByte(0x70);

                        I2C_I2CMasterSendRestart(HMC_REG,
I2C_I2C_WRITE_XFER_MODE);
```

```
                        I2C_I2CMasterWriteByte(0x01);
                        I2C_I2CMasterWriteByte(0x01);

                        I2C_I2CMasterSendRestart(HMC_REG,
I2C_I2C_WRITE_XFER_MODE);

                        I2C_I2CMasterWriteByte(0x02);
                        I2C_I2CMasterWriteByte(0x00);

                        I2C_I2CMasterSendStop();

                        UART_PutString("HMC INIT OK...\r\n");
                        UART_PutCRLF(0);
            }

            else
            {
                        UART_PutString("ERROR INITIALIZING HMC5883l...");
                        UART_PutCRLF(0);
                        I2C_I2CMasterSendStop();
            }

            CyDelay(1000);

            return (bStatus);
}
```

calculate_angles.c

```
#include <calculate_angles.h>

void tilt_calc(sensor *acc_bits, Angles *angles)
{
		float acc_val_x = ((float)(acc_bits->x));
		float acc_val_y = ((float)(acc_bits->y));
		float acc_val_z = ((float)(acc_bits->z));

		float angle_pitch, angle_roll, angle_yaw;

		angle_roll = (atan2f(acc_val_y, acc_val_z)+PI);
		angle_pitch = (atan2f(acc_val_x, acc_val_z)+PI);

		angle_yaw = (atan2f(acc_val_y, acc_val_x));

		angles->roll = (angle_roll);

		angles->pitch = (angle_pitch);
		angles->yaw		= (angle_yaw);

}

// TÄMÄ ALGORTIMI TOIMII VAIN ~45 ASTEEN TILTTIIN

void compensateTilt(sensor *magnet_raw, Angles *acc)
{
		float comp_x, comp_y;

		float acc_roll = (float)(acc->roll);
		float acc_pitch = (float)(acc->pitch);

		float magnet_raw_x = (float)(magnet_raw->x);
		float magnet_raw_y = (float)(magnet_raw->y);
		float magnet_raw_z = (float)(magnet_raw->z);

		comp_x = (magnet_raw_x * cosf(acc_pitch)) + (magnet_raw_z *
sinf(acc_pitch));
		comp_y = (magnet_raw_x * sinf(acc_roll) * sinf(acc_pitch))
+ (magnet_raw_y * cosf(acc_roll)) - (magnet_raw_z * sinf(acc_roll) *
cosf(acc_pitch));

		magnet_raw->x = comp_x;
		magnet_raw->y = comp_y;
}

float rad_to_deg(float rads)
{
		float degrees;

		if(rads < 0)
		{
			rads += (2*PI);
		}

		if (rads > 2*PI)
		{
			rads -= (2*PI);
```

```
            }

            degrees = rads * 180/PI;

            return (degrees);
}
```

Global.h

```
#ifndef _Global_H
#define _Global_H

#include <stdlib.h>
#include <stdio.h>
#include <device.h>

#define RADTODEG                                57.2957795
#define PI                                                  3.14159265
#define ACC_SCALE_FACTOR          4096
#define GYRO_SCALE_FACTOR         65.5
#define MAG_SCALE_FACTOR     0.92f
#define BYTE                                                uint8

typedef struct sensor
{
           int16 x;
           int16 y;
           int16 z;
} sensor;

typedef struct Angles
{
           float yaw;
           float pitch;
           float roll;
} Angles;

extern sensor gyro;
extern sensor acc;
extern sensor magnet;

extern Angles imu_angles;
extern Angles gyro_rates;
extern Angles kalmanized_angles;

extern sensor gyro_offset;
extern sensor acc_offset;
char* itoa(int16 iBuffer, char cBuffer[]);

#endif
```

calculate_angles.h

```
#ifndef _calculate_angles_H
#define _calculate_angles_H

#include <Global.h>
#include <math.h>

void tilt_calc(sensor *acc_bits, Angles *angles);
float rad_to_deg(float rad);
void compensateTilt(sensor *magnet_raw, Angles *acc);

#endif
```

HMC5883l.h

```
#ifndef _HMC5883l_H
#define _HMC5883l_H

#include <Global.h>

void HMC5883l(void);
BYTE init_HMC5883l (void);

#endif
```

Initialization.h

```
#ifndef _Initialization_H
#define _Initialization_H

#include <Global.h>
#include <kalman_Filter.h>
#include <HMC5883l.h>
#include <MPU6050.h>

void sysInit(void);
void init_sensor_values(sensor *init_this);
void init_imu_angle_values(Angles *init_this);
void calibrate_sensors(void);

#endif
```

MPU6050.h

```
#ifndef _MPU6050_H
#define _MPU6050_H

#include <Global.h>

// Gyroscope MPU6050
#define MPU6050_ADDRESS             0x68 // gyro address, binary =
01101000 when AD0 is connected to GND (see schematics)
#define mpu6050_reg_who_am_i        0x75

//register addresses in the same order as in the datasheet
#define mpu6050_reg_config                      0x1A //config register
26
#define mpu6050_reg_gyro_config  0x1B //gyro config
#define mpu6050_reg_accel_config 0x1C //accelerometer config
#define mpu6050_reg_int_pin_config  0x37//interrupt pin
#define mpu6050_reg_int_enable      0x38//enable interrupt pin func-
tionality
#define mpu6050_reg_smprt_div               0x19//sample rate divider


//accelerations in 16-bit 2's complement format
#define    mpu6050_reg_accel_x_msb              0x3B //59
ACCEL_XOUT[15:8]
#define mpu6050_reg_accel_x_lsb      0x3C //60 ACCEL_XOUT[7:0]
#define mpu6050_reg_accel_y_msb      0x3D //61 ACCEL_YOUT[15:8]
#define mpu6050_reg_accel_y_lsb      0x3E //62 ACCEL_YOUT[7:0]
#define mpu6050_reg_accel_z_msb      0x3F //63 ACCEL_ZOUT[15:8]
#define mpu6050_reg_accel_z_lsb      0x40 //64 ACCEL_ZOUT[7:0]

//Temperature in degrees C = (TEMP_OUT Register Value as a signed
quantity)/340 + 36.53
#define mpu6050_reg_temp_msb                0x41 //65 TEMP_OUT[15:8]
#define mpu6050_reg_temp_lsb                0x42 //66 TEMP_OUT[7:0]

//gyro measurements in 16-bit 2's complement format
#define mpu6050_reg_gyro_x_msb       0x43 //67 GYRO_XOUT[15:8]
#define mpu6050_reg_gyro_x_lsb       0x44 //68 GYRO_XOUT[7:0]
#define mpu6050_reg_gyro_y_msb       0x45 //69 GYRO_YOUT[15:8]
#define mpu6050_reg_gyro_y_lsb       0x46 //70 GYRO_YOUT[7:0]
#define    mpu6050_reg_gyro_z_msb               0x47 //71 GY-
RO_ZOUT[15:8]
#define mpu6050_reg_gyro_z_lsb       0x48 //72 GYRO_ZOUT[7:0]

#define mpu6050_reg_pwr_mgmt_1       0x6B
#define mpu6050_reg_pwr_mgmt_2       0x6C

/*Our functions*/
int mpu6050_Start(void);
int mpu6050_read_register(unsigned char *read_here,unsigned char
reg_address,unsigned char n);
int mpu6050_write_register(unsigned char reg_address, unsigned char
writeData);

#endif
```

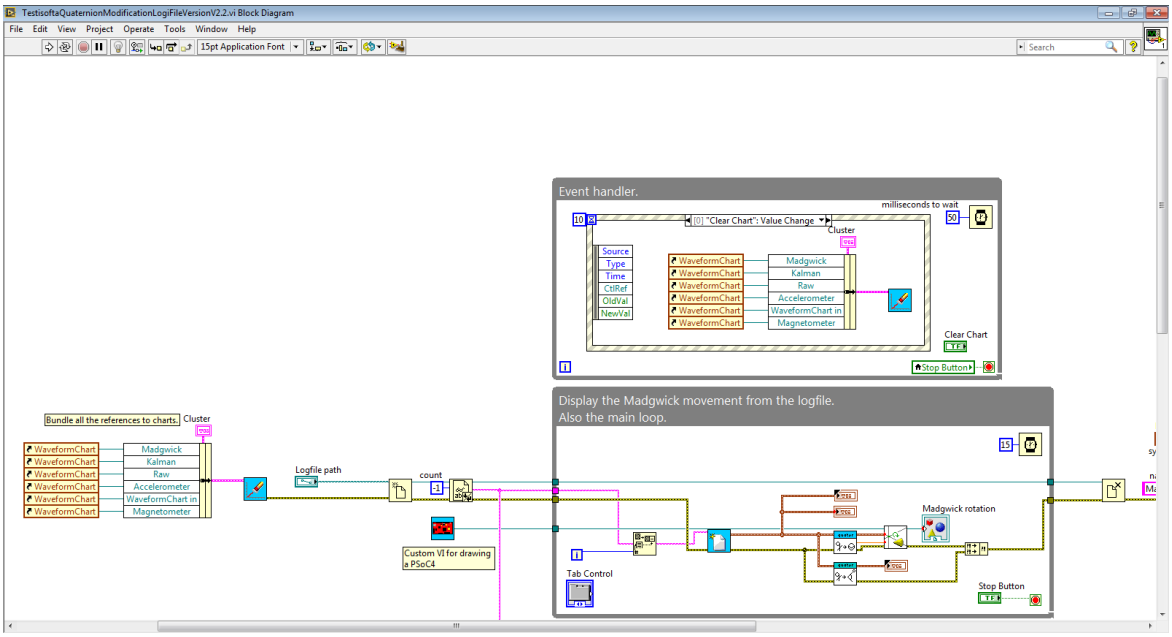
kalman_Filter.h

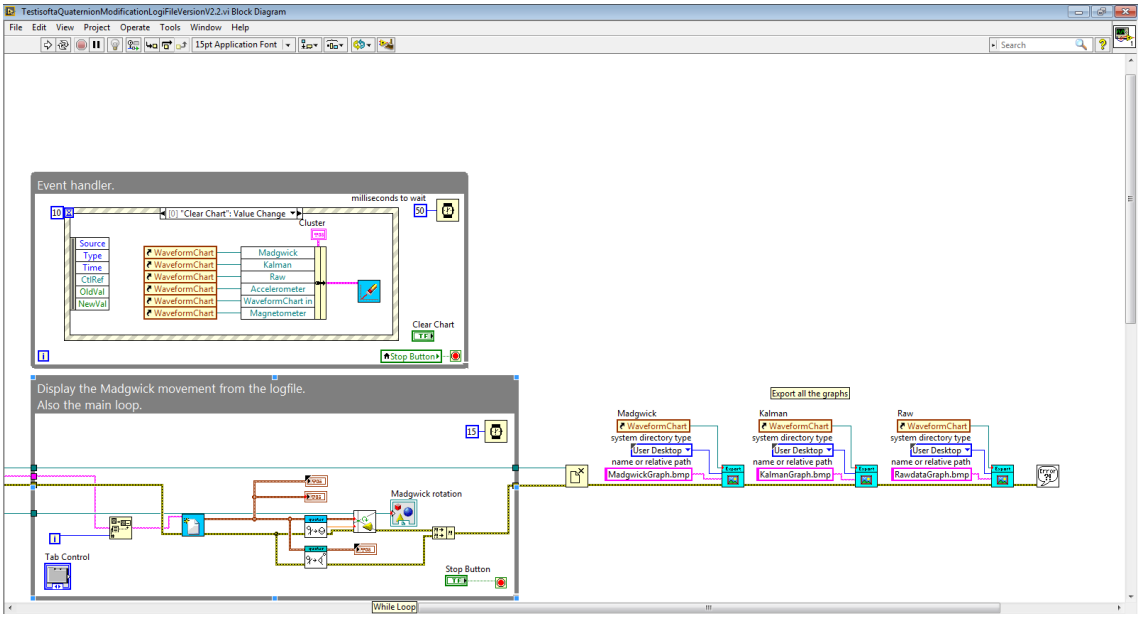
```
#ifndef _kalman_FILTER_H
#define _kalman_FILTER_H

#include <Global.h>

typedef struct kalmanVariables
{
            float Q_angle;        // Q_arvio kiihtyvyysanturin kohinan
aiheuttamasta v��rentym�st�
            float Q_gyroBias;     // Q_gyroBias arvio gyron driftauksen
aiheuttamasta v��rentym�st�
            float R_angle;                  // R_angle
Kiihtyvyysanturin mittauksen kohinan vaihtelu (kovariANSSI)
            float dt;                       // Time step

            float bias;                                 // Gyron
driftaus
            float angle;                    // angle on filtter�it�v�
arvo
            float P[2][2];        //
            float K[2];                     // Kalman Gain factor

} kalmanVariables;

extern kalmanVariables kalman_variables_pitch;
extern kalmanVariables kalman_variables_roll;
extern kalmanVariables kalman_variables_yaw;

void initKalman(kalmanVariables *init_this);
float kalman_filter(float newAngle, float newRate, kalmanVariables
*kVariables);
#endif
```

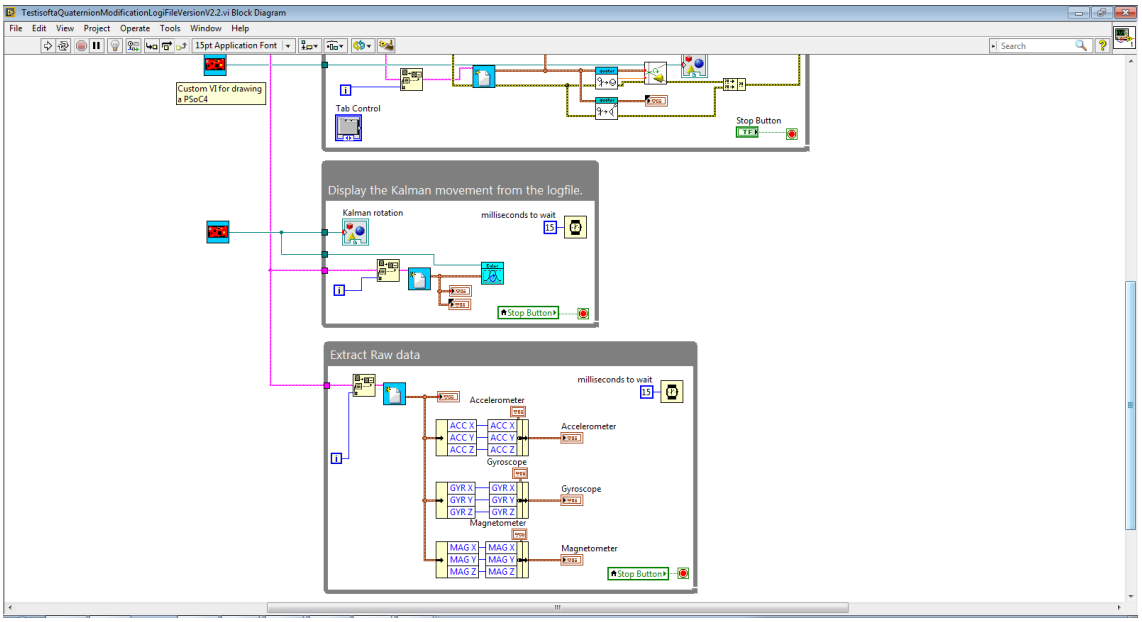
mahony.h

```
#ifndef _mahony_h
#define _mahony_h

#include <Global.h>

//--------------------------------------------------------------------
--------------------------------
// Variable declaration

extern Quaternions q;
extern volatile float twoKp;                          // 2 * propor-
tional gain (Kp)
extern volatile float twoKi;                          // 2 * integral
gain (Ki)

extern volatile float beta;                                    //
algorithm gain
extern volatile float q0, q1, q2, q3;       // quaternion of sensor
frame relative to auxiliary frame
//--------------------------------------------------------------------
--------------------------------
// Function declarations

void MadgwickAHRSupdateIMU(float gx, float gy, float gz, float ax,
float ay, float az);
void MadgwickAHRSupdate(float gx, float gy, float gz, float ax, float
ay, float az, float mx, float my, float mz);
#endif
```

# LabVIEWn block diagramm




TestisoftaQuaternionModificationLogiFileVersionV2.2.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Custom VI for drawing a PSoC4
Tab Control
Stop Button
Display the Kalman movement from the logfile.
Kalman rotation
milliseconds to wait
Stop Button
Extract Raw data
milliseconds to wait
Accelerometer
ACC X
ACC Y
ACC Z
Accelerometer
Gyroscope
GYR X
GYR Y
GYR Z
Gyroscope
Magnetometer
MAG X
MAG Y
MAG Z
Magnetometer
Stop Button